

AD-A087 610

CUNDALL (PETER) ASSOCIATES VIRGINIA WATER (ENGLAND)

F/G 8/7

UDEC - A GENERALISED DISTINCT ELEMENT PROGRAM FOR MODELLING JOI--ETC(U)

MAR 80 P A CUNDALL

DAJA37-79-C-0548

UNCLASSIFIED

PCAR-1-80

NL

1-1

2-1-80

END

DATE

FORMED

9 80

DTIC

ADA 087610

LEVEL

9

UDEC - A GENERALISED DISTINCT
ELEMENT PROGRAM FOR MODELLING
JOINTED ROCK

Final Technical Report

by

Peter Cundall

March 1980

EUROPEAN RESEARCH OFFICE

United States Army

London England

CONTRACT NUMBER DAJA37-79-C-0548

Peter Cundall Associates

DTIC
ELECT
S AUG 6 1980
A

Approved for Public Release; distribution unlimited

DDC FILE COPY

80 8 6 003

Unclassified

R&D 2729

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|--|-------------------------------------|---|
| 1. REPORT NUMBER (14) PCAR-1-80 | 2. GOVT ACCESSION NO. AD-A087610 | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle) (b) UDEC - A GENERALISED DISTINCT ELEMENT PROGRAM FOR MODELLING JOINTED ROCK. | | 5. TYPE OF REPORT & PERIOD COVERED (9) Final Technical Report |
| 6. AUTHOR(s) (10) Peter A. Cundall | | 7. PERFORMING ORG. REPORT NUMBER |
| 8. PERFORMING ORGANIZATION NAME AND ADDRESS Peter Cundall Associates 14 Cabrera Avenue Virginia Water, Surrey, England | | 9. CONTRACT OR GRANT NUMBER(s) (15) DAJA37-79-C-0548 |
| 10. CONTROLLING OFFICE NAME AND ADDRESS Defense Nuclear Agency Washington, DC 20305 | | 11. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 6.11 02A 1T161102BH57-01 (16) |
| 12. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) U. S. Army Research & Standardization Group (Eur) Box 65, FPO New York, NY 09510 (12) 74 | | 13. REPORT DATE March 1980 (11) |
| 14. DISTRIBUTION STATEMENT (of this Report) Approved for Public Release Distribution unlimited | | 15. NUMBER OF PAGES 69 |
| 15. SECURITY CLASS. (of this report) None | | 16. DECLASSIFICATION/DOWNGRADING SCHEDULE |
| 17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) | | |
| 18. SUPPLEMENTARY NOTES | | |
| 19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Rock mechanics; Mathematical models; Rock dynamics ; Crater formation; Jointed rocks | | |
| 20. ABSTRACT (Continue on reverse side if necessary and identify by block number) A new computer program, UDEC, has been developed which simulates the behavior of jointed rock masses subjected to high and transient loadings. It is an explicit, time-marching procedure that models assemblages of discrete blocks or particles that interact mechanically. The program includes modes that describe the behavior of rigid blocks, simple deformability, and full deformability. The modes and other components are also designed to interact so as to model other kinds of physical behavior, such as fluid interaction, edge-to-edge contact, and soft corners. | | |

DD FORM 1 JAN 73 1473

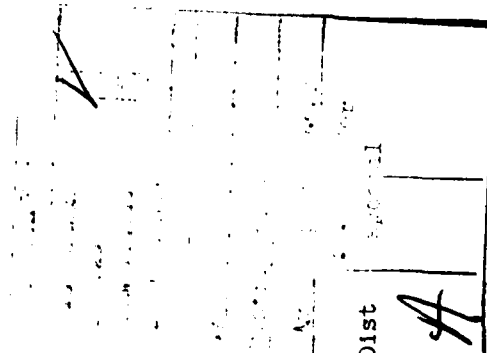
EDITION C NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

| | <u>Page</u> |
|---|-------------|
| List of Figures | i |
| Acknowledgements | ii |
| CHAPTER 1 - INTRODUCTION | 1 |
| CHAPTER 2 - ADDITIONS AND CHANGES TO PREVIOUS WORK | 4 |
| CHAPTER 3 - DATA STRUCTURES | 16 |
| CHAPTER 4 - USE OF PROGRAM | 30 |
| CHAPTER 5 - CONCLUSIONS AND FUTURE DEVELOPMENTS | 33 |
| REFERENCES | 34 |
| APPENDIX I - USER'S MANUAL | 35 |
| * Appendix I-1 Input Commands | 36 |
| * Appendix I-2 Error Messages | 40 |
| APPENDIX II - EXAMPLE RUNS | 41 |
| * Appendix II-1 Ball Rolling | 42 |
| * Appendix II-2 Collapse of Opening in Jointed Rock | 45 |
| APPENDIX III - PROGRAM GUIDE | 50 |
| * Appendix III-1 Main Common Block Variables | 51 |
| * Appendix III-2 Parameters and Data Group | 53 |
| * Appendix III-3 Subroutine Functions | 57 |
| * Appendix III-4 Subroutine Calling Map | 64 |
| APPENDIX IV - GUIDE TO PROGRAM CHANGES | 67 |



LIST OF FIGURES

| | | <u>Page</u> |
|-----------|---|-------------|
| Fig. 1 | Changes in linked lists that occur when two blocks come into contact. | 17 |
| Fig. 2 | Two types of scan that are possible with the same data structure. | 18 |
| Fig. 3 | Linked lists for main data arrays. | 21 |
| Fig. 4 | Block pointers and reverse corner links. | 23 |
| Fig. 5 | Convention used for pointers within a contact array. | 24 |
| Fig. 6 | Structure of 'junk list' holding redundant groups of memory. | 25 |
| Fig. 7 | To show the convention for ordering contact and corner data. | 26 |
| Fig. 8 | Domain Linkages. | 27 |
| Fig. II-1 | System of blocks prior to deletion | 44 |
| Fig. II-2 | Motion of 'block' under gravity. | 44 |
| Fig. II-3 | Initial Geometry | 45 |
| Fig. II-4 | Block Movement Sequence | 47 |
| Fig. II-5 | " " " | 47 |
| Fig. II-6 | " " " | 48 |
| Fig. II-7 | " " " | 48 |
| Fig. II-8 | " " " | 49 |

ACKNOWLEDGEMENTS

The contributions of Dr. Gurdip Kalsi, Ms. Helen Adhami and Ms. Susan Backhouse are gratefully acknowledged. Dr. Kalsi assisted in program development and documentation, and was responsible for testing the program. Ms. Adhami produced the report and Ms. Backhouse supplied the illustrations.

1.0 INTRODUCTION

1.1 Background

The work described in this report follows that performed under contract DACA39-77-C-0004 and presented in "Computer Modelling of Jointed Rock Masses" by Cundall et al (1978)*.

In the previous study several computer programs were developed as examples of the "distinct element method", which is an explicit, time-marching procedure that models assemblies of discrete blocks or particles that interact mechanically. The first program was restricted to rigid blocks, and was a translation into Fortran of an earlier machine-language code that used interactive graphics to create and manipulate blocks. A second program introduced simple deformability, whereby each block has three degrees of freedom to deform internally. A modified version of the rigid block program was written in order to allow blocks to split into two, using a simple criterion based on the applied loads and the block dimensions. Finally, an experimental program was written in which blocks could be discretised internally into finite-difference triangles. Such blocks are termed "fully-deformable".

Apart from the inconvenience in having different facilities available in different programs, the computer programs were written with no particular emphasis on efficiency or flexible data structures, since the intention was to demonstrate and test some new formulations. Furthermore, it has become apparent that it would be difficult to represent certain physical behaviour, such as fluid interaction, edge-to-edge contact and soft corners, without major overhaul of the program logic and data structures.

These considerations prompted the program development described in this report.

1.2 Scope of Present Work

A completely new program, UDEC⁺, has been developed, which provides, in one package, almost all of the capabilities that existed separately in the

* Cundall, P.A. et al, Technical Report N-78-4, U.S. Army Engineers Waterways Experiment Station.

⁺ Universal Distinct Element Code.

previous programs. The principal objective was to write a program that would allow rigid blocks, simply-deformable blocks and fully-deformable blocks to be mixed together in one numerical simulation. It is often useful to mix different types of block for the following reasons: near a free surface in jointed rock, the movements arise predominantly from slip and opening of joints. In these regions rigid blocks may be used, for maximum efficiency of calculation. On moving away from a free surface, towards the interior of a rock mass, joint displacements diminish in comparison with deformations of the intact rock, and the stress distribution is determined largely by the elastic properties of the rock. Deformable blocks should be used in these regions to represent the rock behaviour correctly. The correct boundary conditions are especially important for dynamic calculations in which incident waves are to be propagated towards a rock structure, and reflected waves are to be absorbed. Again, deformable blocks are needed near the boundary to provide the correct propagation velocity and the correct driving impedance for absorbing boundaries.

Although the contractual requirement of the study was limited to the provision of a program incorporating the three types of block noted above, the opportunity was taken to re-examine the way in which the data describing the blocks is represented: the "data structure". The data structure determines to a very great degree how easy it is to represent diverse physical phenomena. Two guiding principles influenced the choice of a data structure: firstly it was assumed that the best data structure is the one that corresponds in a topological sense most closely to the physical structure. In previous programs, the only correspondence was between stored variables and physical variables. The new program actually stores the variables in memory in a way that has identical topological properties to the physical arrangement. The second assumption that was kept in mind during the design of program UDEC was concerned with the way in which computer hardware is developing. Typical memory sizes have increased and costs have decreased by orders of magnitude over the last decade; megabyte memories are now common in minicomputers, and are just becoming available for microcomputers. Execution speeds have also increased, but at a slower rate. Often, when writing a program, it is possible to make a trade-off between memory use and execution time: for example, variables may be saved to avoid recalculation within a loop. The philosophy of reducing execution time at the expense of greater memory requirements has been adopted throughout.

The numerical formulations of the various block types are almost identical to those documented in the previous report by Cundall et al (loc cit). The formulations are not repeated in this report, and the reader is encouraged to read the previous report in conjunction with this one. Any differences, such as rounded corners, are described in Chapter 2, which also documents other changes and additions arising from the new data structure. The data structure itself is documented in detail in Chapter 3.

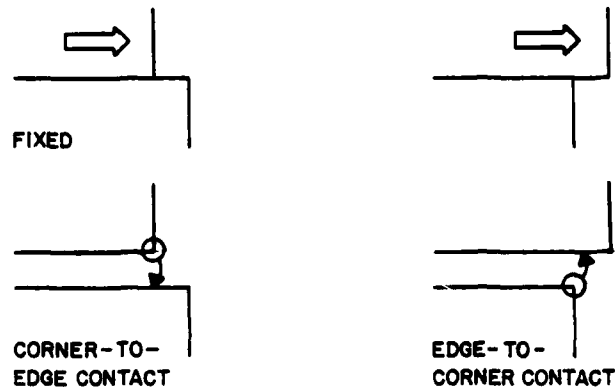
Program UDEC allows interaction between three block types: rigid, simply-deformable and fully-deformable. Of the six possible interactions (e.g. rigid/rigid, rigid/simply-deformable etc.) five require (due to their basic formulation) that contacts shall consist of finite stiffnesses. Only the interaction between fully-deformable blocks has the possibility of rigid contact between opposing grid-points. Rezoning is necessary in this case to keep contacting grid-points opposite one another. It was decided to omit rezoning from program UDEC, and impose a restriction that all contacts be of finite stiffness. In practice, an arbitrarily-large stiffness may be used, but at the expense of a very small time step. It was felt that the complication of rezoning was inappropriate in a program that was primarily intended to model rock blocks with finite-stiffness joints, and that its inherent inefficiency could not be justified by the very small number of cases for which it might be used.

2.0 ADDITIONS AND CHANGES TO PREVIOUS WORK

There are a number of problems associated with existing distinct element programs for modelling assemblies of angular blocks. The more important of these are as follows:

- a. The system of "boxes" used for coarse classification of blocks is quite efficient when the blocks are of a similar size. For blocks that are much larger than the box dimension, much time is taken searching all the boxes overlapping the block and depositing and moving entries among these boxes. For blocks that are small compared to the box size, many potential contacts must be tested within each box, although only a few will be accepted.
- b. It is necessary to perform updates (searches for new contacts) globally in all programs except RBMC, owing to the difficulty of guaranteeing that a given corner will always locate a nearby edge.
- c. There is a limitation on the coordinate system that can be chosen, so that the dimensions of a given block system must be scaled before running a problem. This restriction is made because the magnitudes of block coordinates are used to trigger re-boxing of pointers within the box system.
- d. Blocks cannot be deleted, except in the original machine-language version of the rigid-block program.
- e. "Hang-ups" occur when two blocks overlap by an arbitrarily-small amount, because block corners are assumed to be sharp and infinitely strong.
- f. Contact is always between an edge and a corner; edge-to-edge contact is only approximately represented by two edge-to-corner contacts.

- g. When the edge of one block slides past the edge of another, as shown below, there is a discontinuity in contact force because one edge-to-corner contact must break before another corner-to-edge contact can form. Stored energy is lost because a load-carrying contact is suddenly deleted.



All the problems noted above are eliminated in the new formulation of the distinct element method described in this report. The main changes are to the data structures, and in the fact that rounded corners are used. Several specific innovations are described below.

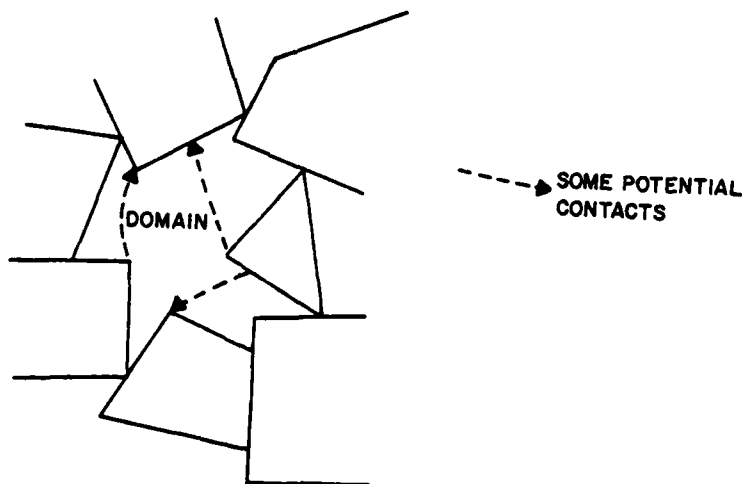
2.1 Detection of Contacts

Chapter 3 describes the new data structures in detail, and the way in which their topological properties correspond closely with those of the physical system of blocks that they represent. It is this correspondence that allows the old "box" scheme to be dispensed with: the connectivity of the physical system is built into the data structure, which means that potential contacts with a given block may be detected by a local examination of the linked-list network surrounding the block. However, for the scheme to work well, there must be a well-developed connectivity; the main application will be to systems of blocks,

each of which is very near several other blocks. This is likely to be true for program UDEC, which has the main applications in modelling jointed rock masses, where blocks are all touching initially.

2.2 Update Triggering

An "update" is defined here as a scan of some region to determine if new contacts should be made or to erase unwanted contacts. As illustrated below, a new contact can only arise physically within a closed region between blocks, called a "domain".



For this reason, an update is limited to an individual domain, and is triggered by significant relative motion occurring within the domain. A scheme has been adopted whereby a fictitious displacement is accumulated for each domain. This displacement is related to the relative motion that has taken place in the domain since the previous update, and is used to trigger the next update when the displacement exceeds a certain tolerance. At each time-step, the greatest relative x- and y-velocities are recorded between any two corners within the domain (including corners that are part of a contact). The fictitious displacement is then accumulated as follows:

$$u^f := u^f + \left\{ \max(|\dot{u}_x^m|, |\dot{u}_y^m|) \right\} \cdot \Delta t,$$

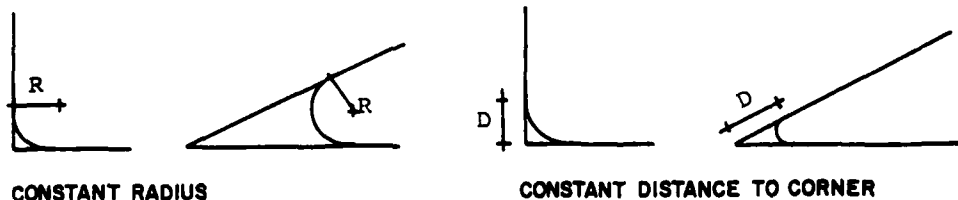
where \dot{u}_x^m and \dot{u}_y^m are the greater relative velocities, u^f is the fictitious displacement and Δt is the time-step. An update is done for the domain when

$$u^f > \frac{T_m}{\sqrt{2}}$$

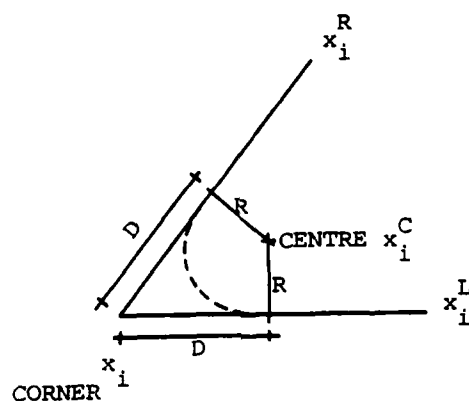
where T_m is the tolerance on making contacts ("contacts" are established before the two blocks actually touch: T_m is the distance between potentially contacting points). The criterion for triggering updates given above ensures that contacts are always detected before physical contact is made.

2.3 Rounded Corners

In program UDEC, blocks behave mechanically as if each corner consists of an arc of a circle; the arc is tangent to the two adjoining edges. The circle is defined by specifying the distance from the block corner to the intersection points of the circular arc with the adjoining edges. This procedure seems physically more reasonable than a specification of a constant radius for all corners, since sharp corners would be considerably truncated if the same radius was used for all corners.



The distance, D , may be specified by the user, and applies to all blocks. It is stored as Fortran variable DTOL. Radius and circle centre are calculated for each corner as follows:



$$Z^L = \left\{ (x_i^L - x_i^L) (x_i^L - x_i^L) \right\}^{1/2}$$

$$Z^R = \left\{ (x_i^R - x_i^R) (x_i^R - x_i^R) \right\}^{1/2}$$

unit vectors:

$$u_i^L = (x_i^L - x_i^L) / Z^L$$

$$u_i^R = (x_i^R - x_i^R) / Z^R$$

tangent unit vectors:

$$t_2^L = u_1^L \quad t_2^R = u_1^R$$

$$t_1^L = -u_2^L \quad t_1^R = -u_2^R$$

by vector addition,

$$Du_i^L + Rt_i^L + Rt_i^R = Du_i^R$$

therefore

$$R = \frac{D(u_1^R - u_1^L)}{t_1^L + t_1^R}$$

or

$$R = \frac{D(u_2^R - u_2^L)}{t_2^L + t_2^R}$$

$$R = \frac{D(u_1^L - u_1^R)}{u_2^L + u_2^R} \quad (1)$$

$$R = \frac{D(u_2^R - u_2^L)}{u_1^L + u_1^R} \quad (2)$$

Either form (1) or form (2) may be used, depending on the magnitude of $u_2^L + u_2^R$ compared to $u_1^L + u_1^R$. The expression with the largest denominator is used in UDEC.

The circle centre is found from:

$$x_i^C = x_i + Du_i^L + Rt_i^L$$

In the present form of UDEC, corners are assumed to be rounded only for the purpose of calculating contact mechanisms; block mass and moment of inertia are calculated on the assumption that corners are angular. This assumption is not essential, and the program can be changed quite easily.

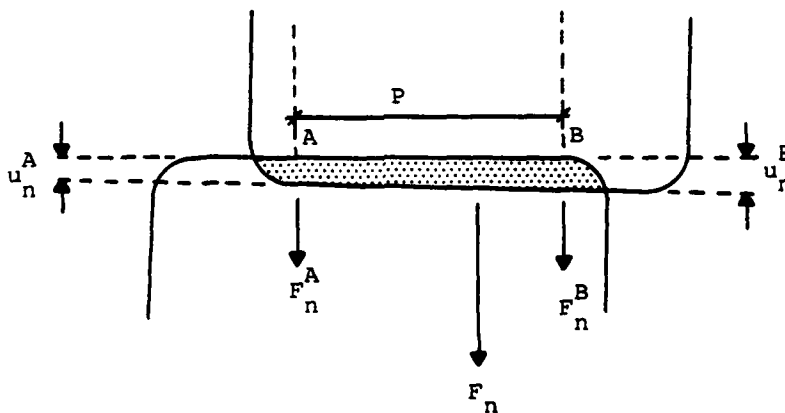
2.4 Edge-to-edge contact

Chapter 3 contains a description of the data structures, and in particular the linked-list representation of a domain. A domain is a closed area bounded by blocks and contacts: the associated linked-list is a circular chain of contacts and corners encountered during an anti-clockwise scan of the domain boundary.

Only two types of contact are needed by the data structure for representing a system of blocks: corner-to-corner contacts and edge-to-corner contacts.

These will be termed "numerical contacts". Physically, however, edge-to-edge contact is important because it corresponds to the case of a rock joint closed along its whole length. A physical edge-to-edge contact corresponds to a domain with exactly two numerical contacts in its linked-list. When an edge-to-edge contact is recognised in this way during a domain scan, it is treated differently from other contacts in respect of its physical behaviour. For example, it is more appropriate to express the constitutive model for such a contact in terms of stresses rather than forces. Stresses can be evaluated by the program since the length of the physical contact is known from the distance between the two numerical contacts.

Many types of constitutive model for edge-to-edge contact may be contemplated. The program provides the displacements at either end of the joint, and the model must furnish the average normal and shear stress, and the line of action of the resultant forces. The simplest constitutive model is as follows:



$$\Delta \sigma_n = f_n(\sigma_n, \Delta u_n, \Delta u_s)$$

$$\text{where: } \Delta u_n = \frac{1}{2} (\Delta u_n^A + \Delta u_n^B)$$

$$\Delta \tau = f_s(\sigma_n, \tau', \Delta u_n, \Delta u_s)$$

$$\Delta u_s = \frac{1}{2} (\Delta u_s^A + \Delta u_s^B)$$

$$\sigma_n' = \sigma_n + \Delta \sigma_n$$

$\Delta u_n^A, \Delta u_s^A$ = normal and shear displacement increments at end A of joint;

$$\tau' = \tau + \Delta \tau$$

$\Delta u_n^B, \Delta u_s^B$ = same, at end B of joint

σ_n = normal stress

τ = shear stress

prime (') denotes new value.

f_n, f_s = non linear functions

$$F_n = p \cdot \sigma_n'$$

p = joint length

$$F_s = p \cdot \tau'$$

F_n, F_s = overall normal and shear forces

$$F_n^A = r F_n \quad F_s^A = r F_s$$

$$r = \frac{u_n^A}{u_n^A + u_n^B}$$

$$F_n^B = (1-r) F_n \quad F_s^B = (1-r) F_s$$

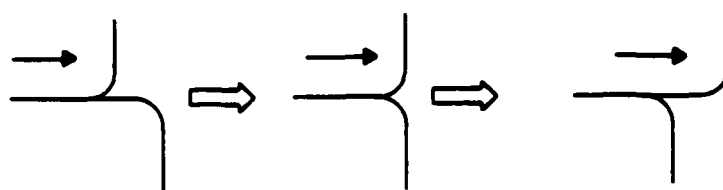
In the procedure presented above, the line of action of the resultant forces is determined by making the ratio of end forces equal to the ratio of end displacements:

$$\frac{F_n^A}{F_n^B} = \frac{u_n^A}{u_n^B}, \quad \frac{F_s^A}{F_s^B} = \frac{u_n^A}{u_n^B}$$

Any other scheme may be substituted: for example, the angle between the two block faces may be used to determine the line of action, or the centroid of the overlap area may be used. In both cases, the program already provides the necessary geometrical data.

2.5 Contact re-classification

One advantage of rounded corners is that there are no abrupt transitions as blocks in contact slide past one another. In the sequence shown below, the same contact is re-classified successively as corner/edge, corner/corner, edge/corner.



In previous block programs, the same sequence would involve the initial contact breaking, and a new one being created. The data structures of UDEC allow the same contact to be retained through the whole sequence. The only change is that a code number denoting the classification of the contact is updated as the blocks move.

2.6 Calculation sequence

In all explicit, time-marching schemes the main calculation cycle consists of applying the law of motion to all mass-points followed by the calculation of force increments from displacement increments for all spring-like elements (contacts, continuum zones, fluid cells, etc.). Program UDEC follows this general scheme although it is complicated by the need to make and break contacts, and the fact that many different types of element are allowed to interact.

For all blocks:

- . accelerate centroids from force-sums.
- . calculate strain-rates for simply-deformable blocks from applied stresses.
- . accelerate grid-point masses from internal and boundary forces for fully-deformable blocks.
- . update corner velocities and displacements.
- . apply new relative velocities to surrounding contacts.
- . reset force-sums to zero.

For all domains:

- . accumulate fictitious domain displacements and update domain if necessary (i.e. make and break contacts).
- . compute incremental pore pressures from velocities around domain; apply resulting forces to blocks.

For all contacts:

- . update contact forces from known relative contact velocities using constitutive model.
- . accumulate centroid force-sums, grid-point force-sums and applied stresses for simply-deformable blocks from contact forces.
- . allow fluid flow between the 2 domains on either side of contact: update pore pressures.

For all zones:

- . compute strain rates; hence new stresses; hence grid-point forces.

2.7 Creation of blocks and joints by splitting

The main application of program UDEC will be to model in-situ rock masses. It is convenient to generate the numerical system of rock blocks by specifying the joints (discontinuities) rather than individual blocks. Not only is the input data less voluminous, but the joint properties can be specified independently of the rock properties. Consequently the principal way of creating a system of rock blocks in UDEC is by splitting existing blocks, and deleting blocks that are not needed. Almost any geometrical arrangement may be created in this way. The subroutine that splits blocks may also be called dynamically, while UDEC is running, so that blocks may fracture as a function of the load applied to them.

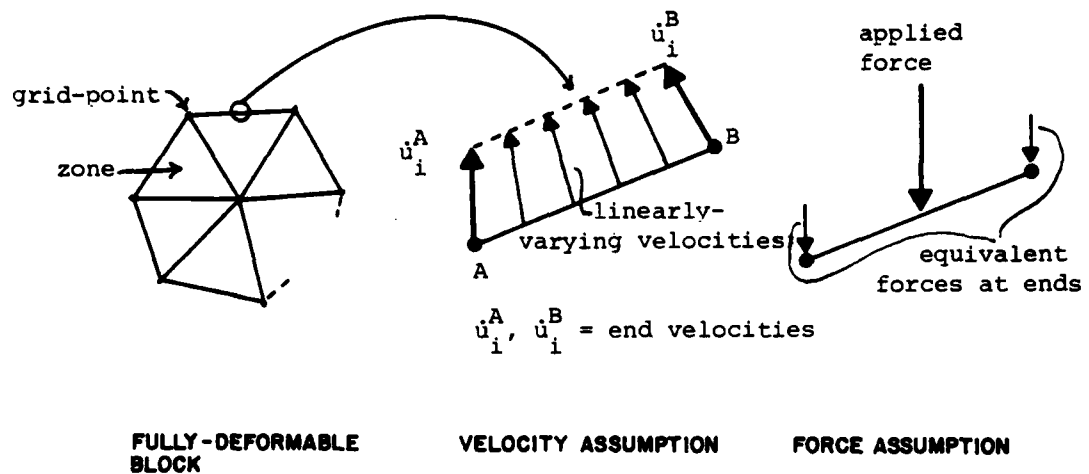
2.8 Pore-pressure calculations

The framework for representing pore-pressure generation and dissipation exists in UDEC but no tests of the capability have been made at the time of writing. The data structure provides a network of inter-connected domains and contacts that can be regarded as reservoirs and pipes respectively. For each domain, increments in fluid pressure can be calculated from the known incremental displacements of the block corners that constitute the boundary to the domain. The block forces arising from the fluid pressures can be calculated, since the coordinates of each point around a domain are known. During a contact scan, the pressures on either side of each contact can be relaxed, by assuming that the constriction corresponding to the contact has a certain hydraulic conductivity. In this way pore-pressures may be continuously generated and dissipated as the blocks move.

2.9 Fully-deformable blocks

As discussed in the Introduction, no re-zoning is performed when fully-deformable blocks slide over one another. The use of finite contact stiffnesses between blocks enables the calculations for adjacent blocks to be decoupled; they communicate through a common boundary force. The mass-points (also called grid-points) on the boundary of a fully-deformable block are accelerated in the normal way during each time step in proportion to the sum of the forces arising both from within the block and from any contacts that exist at the points. Relative contact velocities are updated from the velocities of the block boundaries on either side of each contact.

The general scheme outlined above is complicated by the fact that grid-points from opposing blocks need not coincide: edge-to-corner contact is allowed. In order to calculate forces and displacements at boundary locations other than grid-points, it is assumed that each boundary segment between grid-points acts as a rigid bar with prescribed end velocities: the velocities are assumed to vary linearly along the bar. This assumption is consistent with the assumption that the triangular zones are of the constant-strain type. Any forces acting on the bar are distributed to the two ends, while maintaining moment and force equilibrium.



A further complication is caused by the rounded corners. In the case of rigid and simply-deformable blocks, forces and velocities are transmitted exactly at the point of contact. For fully-deformable blocks, the calculation to do this becomes lengthy, and in some cases ambiguous when several zones meet at the same grid-point. The present version of UDEC contains an approximate, but rapid calculation when transmitting forces and velocities at corners that form part of a contact: the contact force is assumed to act at the corner of the block, and not at the true contact point found by assuming rounded corners. Similarly the velocity transmitted to the contact is taken as the corner velocity even though contact is located somewhat inside the corner-point. The rounded corners still operate as intended, since the contact normals and the relative block positions for contact are correct, but the slight error in location of contact forces and velocities introduces a moment and rotation error, respectively. It will be necessary to gain experience using the program to find out whether the errors are significant in practice: if they are too large, the program can be modified, but at the expense of efficiency.

3.0 DATA STRUCTURES

3.1 Main Structure

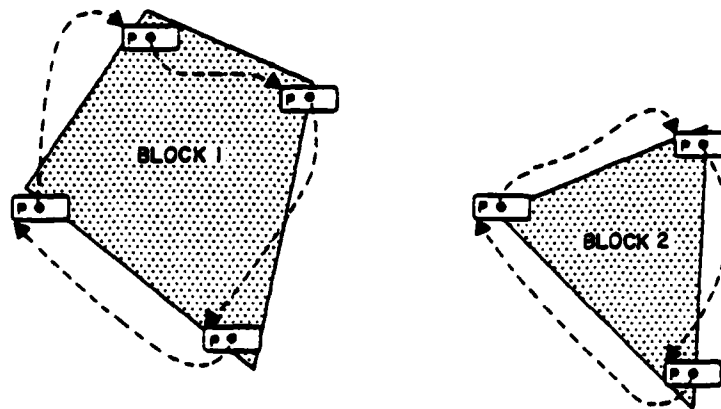
There are many subsidiary pointers and lists that help in retrieving data quickly, but the main data structure is described as follows. Each block has a circular linked-list that corresponds to its boundary. Each corner on the boundary is represented by an array of words that stores the coordinates of the corner, the velocity and several other items. The corner arrays are linked together consecutively in the clockwise direction. When two blocks come into contact their corner lists are broken at the points of contact and an array of words (corresponding to a contact) is inserted into the break, such that the contact array is common to the corner lists of both blocks. The process of making a contact is illustrated in Fig. 1. It should be noted that in all diagrams, computer variables and symbols, the following letters are used in referring to the various entities:

| | | | |
|---|---------|---|------------|
| C | contact | D | domain |
| P | corner | Z | zone |
| B | block | G | grid-point |

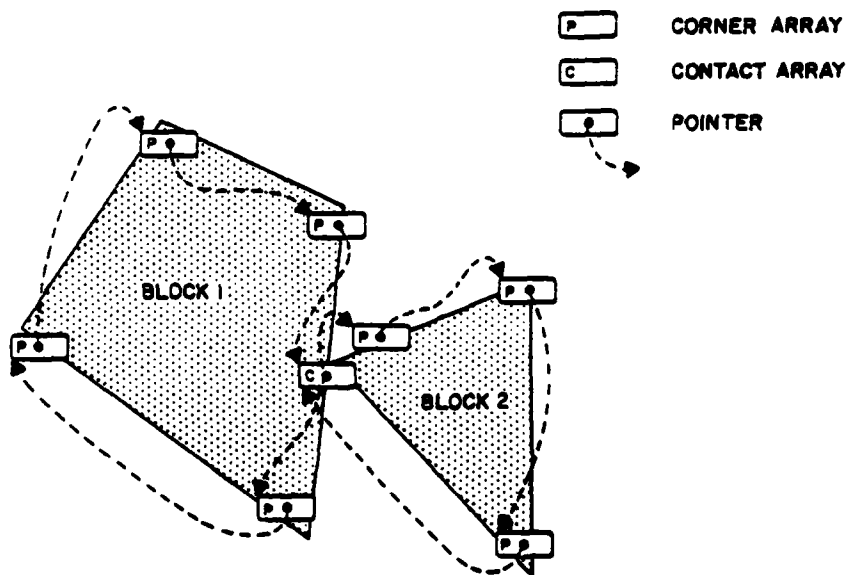
The small boxes labelled "corner array" and "contact array" are placed on the diagram near the physical locations that they represent, but in the program UDEC they exist as groups of contiguous memory addresses contained in the single Fortran array IA(). The actual locations of the groups in IA() are unimportant and arbitrary, since the data in the groups are retrieved by following the appropriate pointers.

The example shown in Fig. 1 illustrates that there are two paths leading away from a single contact. Depending on the scanning strategy used, it is possible to follow either block boundaries or the spaces (domains) between blocks. Fig. 2 illustrates the two possibilities for the same data structure.

Fig. 2(a) shows an anti-clockwise scan that traces the boundary of an inter-block domain; Fig. 2(b) shows a clockwise scan that traces a block boundary. The first type of scan is useful when calculating pore pressures from changes in pore volumes, while the second type of scan is used when up-dating the velocities of corners and contacts around a block from the motion of the block itself.

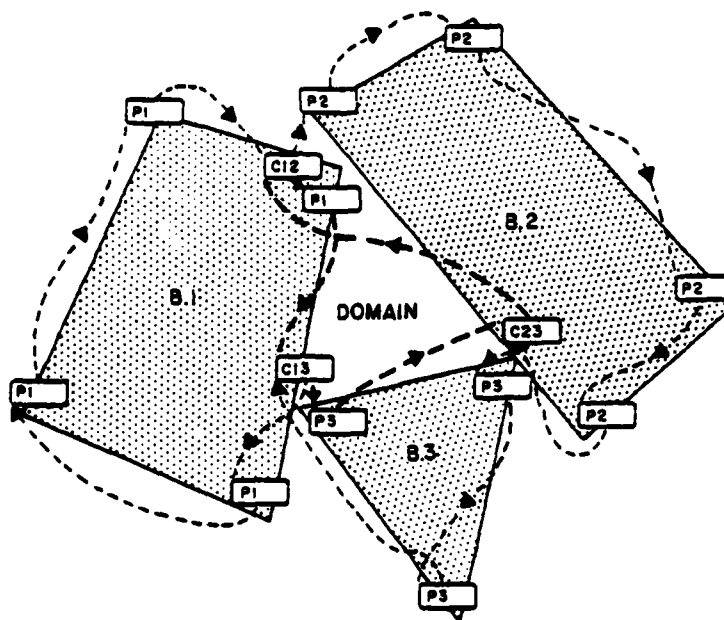


a) BEFORE CONTACT




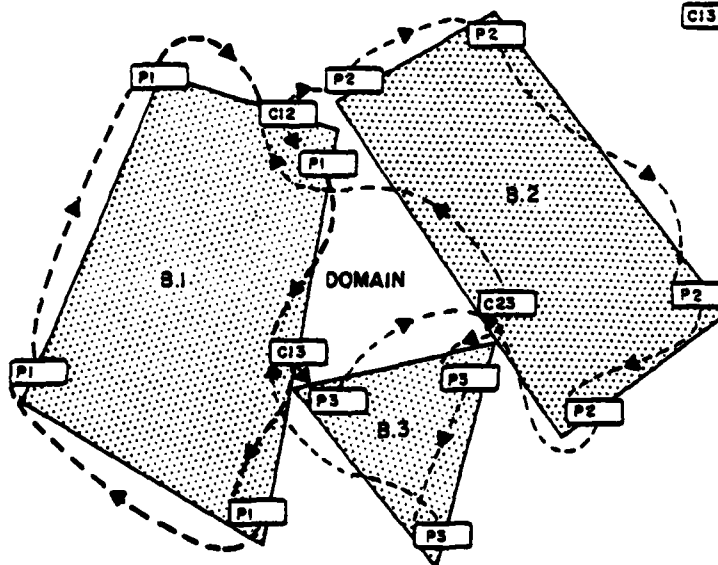
b) AFTER CONTACT

FIG. 1: CHANGES IN LINKED LISTS THAT OCCUR WHEN TWO BLOCKS COME INTO CONTACT



a)  SCAN AROUND THE DOMAIN BETWEEN BLOCKS

 CORNER OF BLOCK 2
 CONTACT BETWEEN BLOCKS 1 AND 3



b)  EXAMPLE OF SCAN AROUND A BLOCK

FIG. 2: TWO TYPES OF SCAN THAT ARE POSSIBLE WITH THE SAME DATA STRUCTURE

PETER CUNDALL ASSOCIATES

The data structure outlined above is better than those used in previous distinct element programs for the following reasons:

- a) each block has direct access to all its contacts, and thereby to all neighbouring blocks; the creation and deletion of contacts is made easy;
- b) closed domains between blocks are defined without additional computing overhead; pore-pressure increments may be calculated from known velocities around the domain boundaries;
- c) each contact has direct access to the two domains on either side of it; hence the calculation of fluid flow between domains is made easy.
- d) edge-to-edge contact may be detected simply by noting when a domain volume collapses to zero.
- e) The search for new contacts can be triggered locally, by monitoring relative displacements within each domain; the part of a block that intrudes into a domain can only touch another block in the domain.

The observations made above support the contention made in the Introduction that the best data representation is that which corresponds most closely to the physical structure: any changes and interactions that occur physically can be treated in exactly the same way in computer memory.

However, the data structure described above is not suitable for representing assemblies of particles that are widely-separated, or particles that move with high velocities. The linked-list scheme assumes that the connectivity of the system changes gradually, and that the domains between particles are clearly defined by a closed loop of contacts. The scheme can handle isolated cases where individual particles lose contact with their neighbours, by using "virtual contacts", which are fictitious links between particles. But the computer program is not designed for multiple virtual contacts; in this case it becomes inefficient.

3.2 Data arrays

The main data structure described above involves two types of data array: corner and contact arrays. Program UDEC uses a number of other arrays (or groups of contiguous memory). The complete set is as follows:

1. Block
2. Domain
3. Corner
4. Contact
5. SDEF extension
6. Zone
7. Grid-point
8. Domain extension

Appendix III lists the complete contents of these data groups. Groups 3 and 4 have already been described; Group 1 stores data for each block, such as centroid, velocity, material type, and constitutive type number. Group 2 stores the pore-pressure for a domain and an accumulated displacement that is used to trigger a contact update for the domain. Group 5 contains the data necessary for simply-deformable blocks, such as stresses and strain-rates. This group is linked to the corresponding block data (Group 1) by an extension pointer that is zero in the case of a rigid block. Groups 6 and 7 store geometrical and connection data for fully-deformable blocks, where the blocks are discretised into triangular zones and grid-points. Again, the linkage from the corresponding Group 1 is via the extension pointer. Group 8 stores data when edge-to-edge contacts are detected. Each Group 8 data block is linked to the corresponding domain data (Group 2) via an extension pointer.

3.3 Support Structures

A number of other lists and pointers are used, so that data can be accessed rapidly and conveniently, particularly when cycling through the main calculation loop.

All blocks, domains, contacts, zones, and grid-points are linked together into five separate lists. Pointers to the starting address of each list are provided; Fig. 3 shows this arrangement schematically.

In order to scan through block corners rapidly, and to access, conveniently, the corners to either side of a given corner, a "reverse list" is provided for each block that links the corners of a block together in the anti-clockwise direction. It will be recalled that the normal clockwise list of

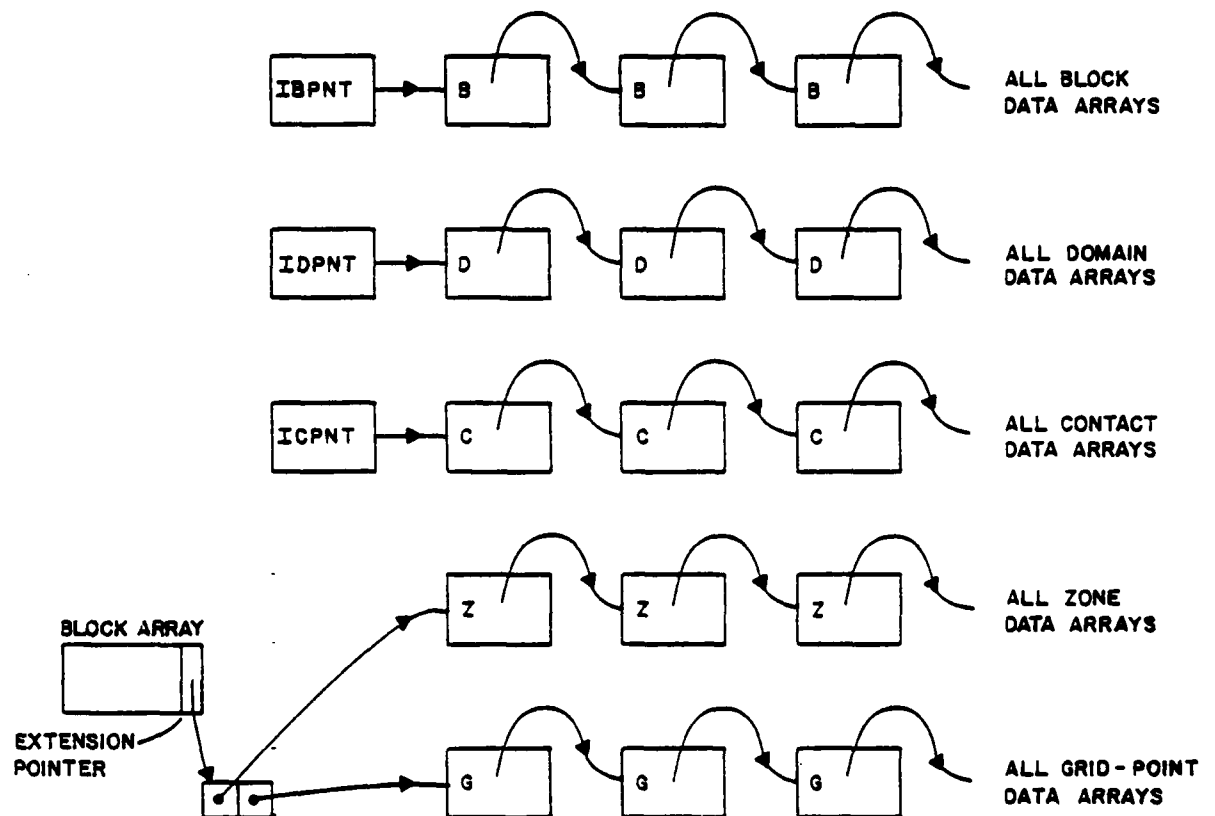


FIG. 3 : LINKED LISTS FOR MAIN DATA ARRAYS

corners could be broken at any number of places to include a contact array. The reverse list has no such interruptions: it simply links corners. Fig. 4 illustrates the reverse list.

Each contact joins two blocks and two domains; pointers are provided in the contact array to these four addresses. Two further words in the contact array point to the two corner lists circulating around the two blocks. It is important to establish an ordering convention for contact pointers, as several exits are possible after entering a contact during a scan. Fig. 5 gives this convention. For example, when entering from Block 1 (with block number equal to the contents of the word with offset KB1), the domain to the left will be found from the word with offset KD1.

A further convention is observed in the ordering of contacts and corner data in the anti-clockwise list running around a block boundary. All contacts appear in the list in the order in which they occur physically. Since corners are rounded, the actual location of a corner is ambiguous, particularly if there are several contacts on one corner. The convention is adopted whereby the corner data array is always placed after all the contact data arrays for that corner. Fig. 7 illustrates this convention. The convention is convenient because a given contact can always locate its associated corner(s) by following the linked list until a corner is found.

Several data arrays contain additional pointers not noted above: a complete list is given in Appendix III for each data type. As an example, each group of corner data for a fully-deformable block includes a pointer to the data group of the corresponding grid-point. Conversely, the grid-point data incorporates a reverse link to the corner data.

3.4 Memory Management

All requests for memory are handled by the subroutine FIND. FIND is given the number of words required, and responds by returning the address of a newly allocated data group, if memory is available. Fresh memory is used to provide space for a new array unless a group with the correct number of words has previously been returned. This reclamation of memory is made possible by maintaining a linked list of redundant memory groups. The list is constructed by subroutine LOSE and is pointed to by variable JUNK, and its structure is illustrated in Fig. 6.

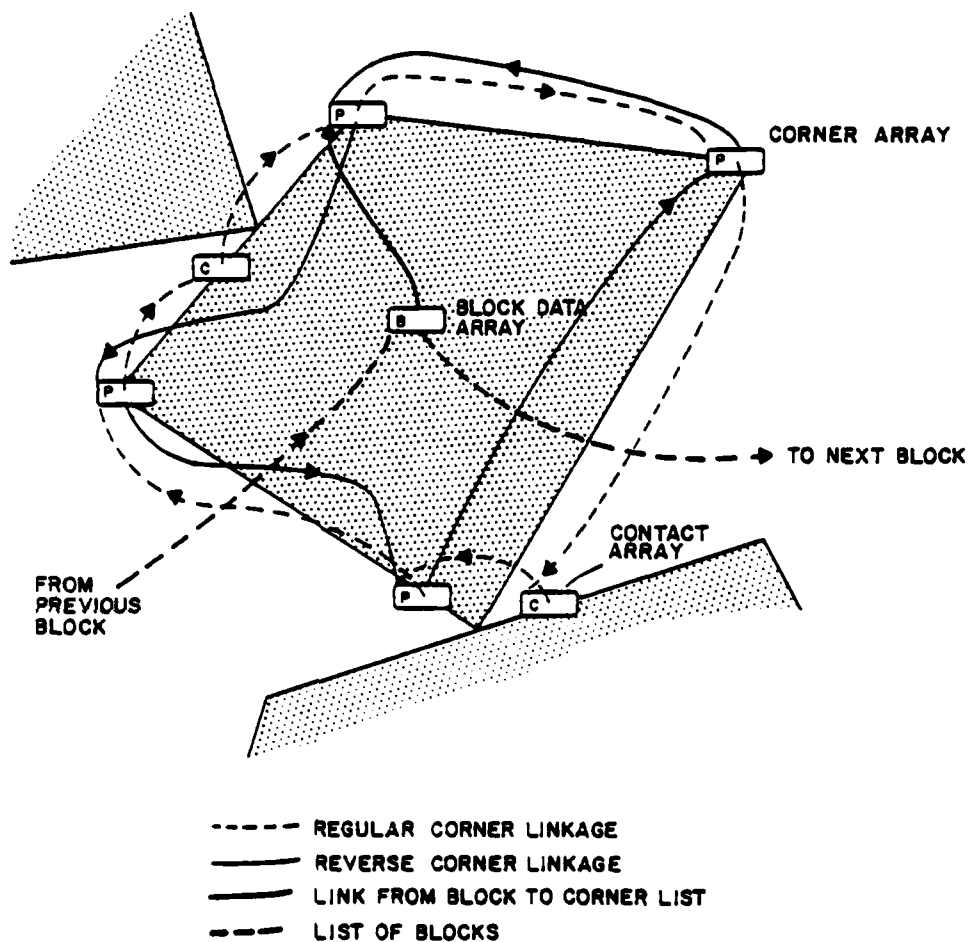
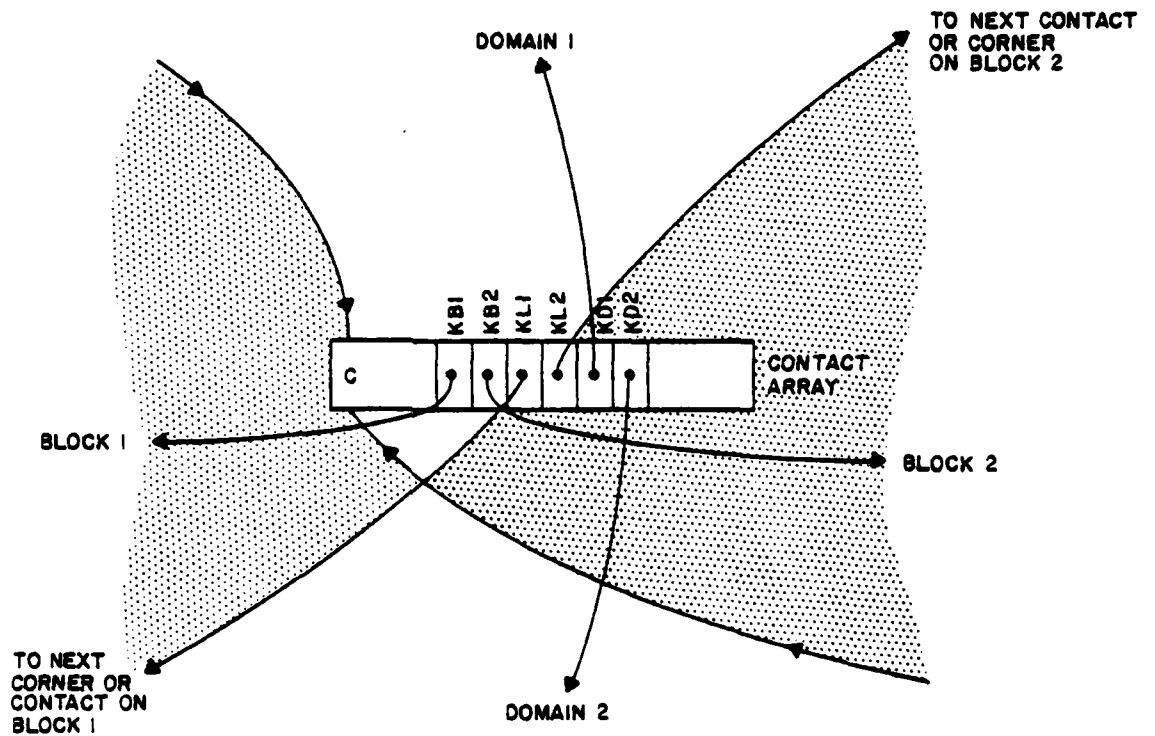
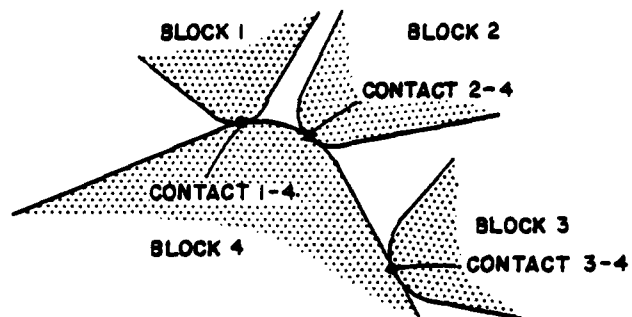


FIG. 4 : BLOCK POINTERS AND REVERSE CORNER LINKS

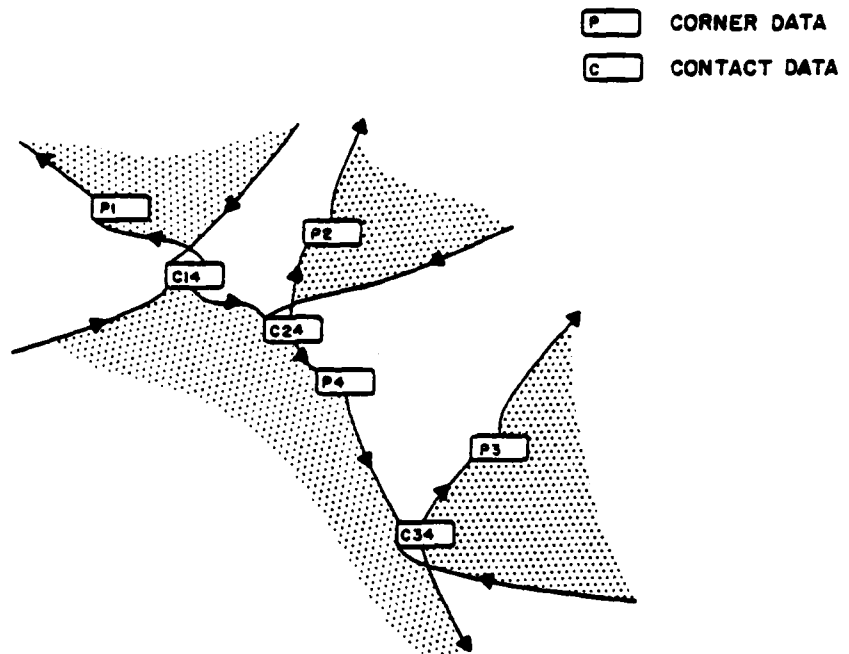


NOTE : KB1, KB2 — KD2 REFER TO THE OFFSETS LISTED IN APPENDIX III

FIG. 5.: CONVENTION USED FOR POINTERS WITHIN A CONTACT ARRAY



a) PHYSICAL ARRANGEMENT



b) REPRESENTATION BY LINKED-LISTS

FIG. 7: TO SHOW THE CONVENTION FOR ORDERING
CONTACT AND CORNER DATA

PETER CUNDALL ASSOCIATES

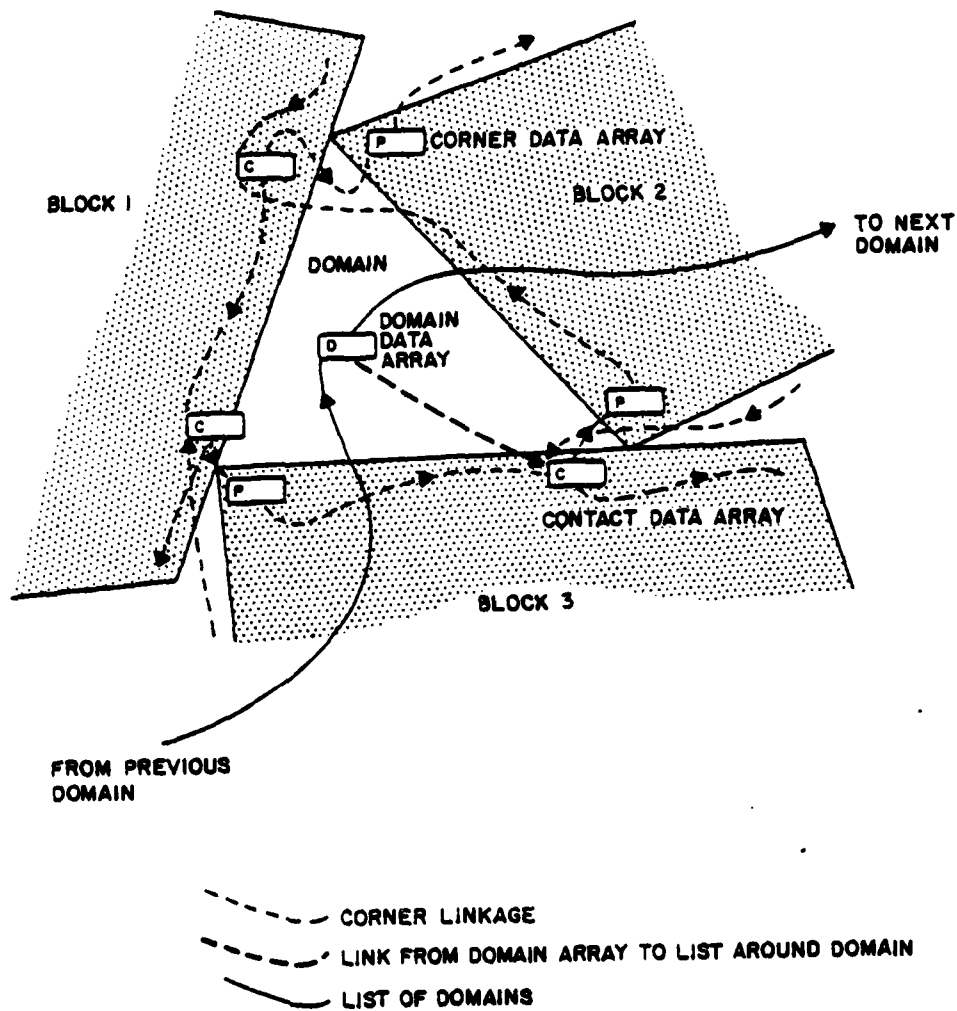


FIG. 8. : DOMAIN LINKAGES

No attempt is made to subdivide memory groups when smaller numbers of words are required, but this technique could be used if desired. The implementation of a suitable scheme should not be too difficult, since all memory management is handled through only two routines.

3.5 Access to data in the main array

Integers and real numbers are both stored in the main array, IA(). In order to save space on small computers, the program UDEC has been written in such a way that the word length for integers can differ from the word length for real numbers. The following schemes are possible, for example:

| <u>Integers</u> | <u>Reals</u> |
|-----------------|--------------|
| 16 bits | 32 bits |
| 32 bits | 32 bits |
| 16 bits | 64 bits |
| 32 bits | 64 bits |
| 64 bits | 64 bits |

This flexibility has been achieved by adopting a particular convention when accessing data from the main array. Integers are accessed directly, as follows:

```
N = IA(IADDR)
```

Real numbers are always accessed via a subroutine call:

```
CALL SUB(IA(IADDR))
```

```
SUBROUTINE SUB (A)
```

```
COMMON/ / KOFF
```

```
DIMENSION A(1)
```

```
R = A(KOFF)
```

Here R is the required real number and KOFF is the offset between the calling address IADDR and the location of the number R. IADDR is typically the pointer to the start of a particular data array, and KOFF is the offset corresponding to the particular variable being accessed.

As set-up at present, the program takes the space of two integers to store one real number. This can be changed by giving different values to the data offsets, which are variables starting with the letter K. Appendix IV gives further details.

4.0 USE OF PROGRAM

The operation of program UDEC is very similar to that of the previous programs RBM and SDEM, but the user has, in general, more flexibility in the sequence of operations that may be requested. Appendix I lists the input commands and provides a description of each. Appendix II contains the input and output from some complete runs. The runs illustrate the use of all available commands, as well as providing sample problems, which can be used to check versions of UDEC set up on other computers.

4.1 Creation of Blocks

The creation of the assembly of blocks differs greatly from that of previous programs. Instead of specifying each block individually, a single, large block is created and then subdivided repeatedly into many small blocks. Blocks may be deleted at any stage in order to generate assemblies of complex geometry. Blocks may be split and deleted even after cycling has started.

4.2 Block Types

Blocks created initially are, by default, rigid. Individual blocks or groups of blocks subsequently may be changed, before running, to simply-deformable or fully-deformable blocks. The program also allows block types to be changed during a run, but this should not be done unless there is a good physical reason for it; the program may have to be modified if stresses, for example, must be preserved.

Fully-deformable blocks are subdivided internally into a mesh of triangular zones. Each zone is specified manually, using a GENERATE command, which takes as arguments lists of grid-points and zones. A grid-point must be assigned to every boundary point (corner) of the block. Grid-points that are placed on edges cause new corners to be created.

4.3 Material and Constitutive Numbers

Each block and contact carries both a material number and a constitutive number. Each material number may be associated with a different set of material properties. Independently, the program may refer to different constitutive models for different blocks or joints. The present version of the program contains block and contact subroutines corresponding to constitutive number 1. These subroutines

model linear elasticity and Coulomb friction, respectively. Dummy subroutines exist for constitutive numbers 2 to 5; these may be replaced by user-written routines as explained in Appendix IV. If a contact is not given a material and constitutive number, it assumes, by default the numbers of one of the blocks comprising the contact. Material properties and even constitutive numbers may be changed during a run, but the user should be sure that the change is physically reasonable. The program prevents the user changing block masses after cycling has commenced, as this would almost certainly be unreasonable.

4.4 Corner Rounding

The ROUND command influences the amount by which corners are rounded, by setting the distance from the true block corner to the point at which the circle is tangent to either edge. The command should not be used after cycling has started because a change in corner geometry would have an unpredictable effect on contact forces. The rounding distance is also used for other purposes, notably for controlling contact detection and deletion. Tolerances are set for these functions, and in the present version of UDEC are taken as fractions of the rounding length. This calculation may be changed by modifying subroutine TOLSET; such a modification may be necessary if very small rounding lengths are needed, since in that case contact updating may be unacceptably frequent.

4.5 Printout

All input lines are echoed, and preceded by the symbol > to distinguish them from output produced by the program.

Printout is generally self-explanatory, with full error messages.

4.6 Restrictions and Cautions

The program cannot be regarded as being in its final form, as several planned facilities, traps and options were not completed in the time available. The program is potentially very powerful, as it can model anything from a continuum to a complete discontinuum. But this very generality ensures that there are many opportunities for misuse.

The calculated critical time-step is only approximate. If numerical instability is suspected, the same run should be made with half the time-step

and double the number of cycles. Any significant difference in the results indicates that the original time-step was too large.

Blocks that become detached from their neighbours may, under some circumstances, not make correct contact again. The program always maintains one "virtual" contact for a block that becomes detached; this is to keep the block linked to the data structure. However a potentially new contact that would cross the track of the virtual contact will not be made correctly, if at all. The logic to deal with this situation is quite straightforward, but has not been written yet. No problems should occur for fairly tight rock masses.

At present, corner radii are only re-calculated when a CYCLE command is given, although the centre coordinates for each corner circle are updated at each time-step. When running problems in which deformable blocks are changing shape rapidly, the total number of required cycles should be split up by using several CYCLE commands. In this way the radii will never be too much in error.

4.7 Incompressible Plastic Flow

Fully-deformable blocks are discretised internally into a mesh of finite-difference triangles. Such assemblies of constant-strain triangles are found to be too stiff when plastic flow is occurring under conditions of near-incompressibility: for example, collapse loads are overestimated. Nagtegaal et al (1974) explained this phenomenon, and Marti and Cundall (1980) proposed a procedure for overcoming the problem. This procedure is called "mixed discretisation" and consists in averaging the volumetric calculation over two adjacent triangles, while the deviatoric calculation is done separately for each triangle. Mixed discretisation could be incorporated easily into UDEC, since a linked list already exists that could serve to combine alternate triangles volumetrically.

5.0 CONCLUSIONS AND FUTURE DEVELOPMENTS

A powerful computer program has been developed that can model a wide spectrum of problems in solid-body mechanics, ranging from a continuum at one extreme to a completely discontinuous medium at the other. Furthermore, arbitrary mixtures of the two can be accommodated. The program differs from previous programs mainly in respect of its data structure, which is designed to have the same topological properties as the physical structure that it represents. The advantage in this is that any structure or connection that exists physically has an analogue in the linked-list space of the data structure.

The contractual objective of the work reported here was to develop a program in which rigid blocks, simply-deformable blocks and fully-deformable blocks could interact with one another. This was achieved. Innovations that have been made, in addition to the new data structure mentioned above, are: the creation of block systems by splitting and deletion, and the use of rounded corners to prevent "locking-up". Parametric studies may be made in which the rounding dimension is varied for the same problem. More utilitarian improvements include free-format input with powerful parameter handling and continuation line logic, and the fact that modifications and changes to blocks and properties may be made at any stage during a run.

More time was spent developing and coding the new data structure than was anticipated. Consequently several planned facilities have been provided for but only exist in skeleton form at present. The reason for this is simply that time ran out, and not that any difficulty is involved. The program UDEC contains incomplete coding for: pore-pressure generation and dissipation, edge/edge contact, automatic zoning for fully-deformable blocks, dynamic cracking, structural connection and handling of initially-free blocks. Some of the coding is almost complete, but in other cases has only just been started. The total time for completion of all items noted above would be about six weeks.

Program UDEC has been written in modular form in anticipation of future extension. Further developments, such as dynamic input and non-reflecting boundaries should present little difficulty.

REFERENCES

NAGTEGAAL, J.C., D.M. Parks & J.R. Rice, (1974), On Numerically Accurate Finite Element Solutions in the Fully Plastic Range, Computer Meth. in Appl. Mech & Eng, 4, 153-177.

MARTI, J. & P.A. Cundall, (1980), Mixed Discretisation Procedure for Accurate Solution of Plasticity Problems", submitted to Int. J. Of Num. & Anal. Meth. in Geomechanics; pre-prints available from: Peter Cundall Associates, 14 Cabrera Avenue, Virginia Water, Surrey, England.

APPENDIX I - USER'S MANUAL

I-1 Input Commands

I-2 Error Messages

Appendix I-1

Input commands for UDEC

Notes: Upper-case letters in a command or parameter must be typed; the remaining letters are optional. Lower-case parameters stand for numeric values. Integers must be given for parameters starting with i,j,k,l,m,n. Real numbers may be given as integers, but not vice versa. Input is free-format: parameters may be separated by any number of the following characters, in addition to spaces:
= () , /

An additional line should be given at the end of the input file (after the STOP command).
The first command should be START or RESTART.

* = comment line
+ = continuation line

| Block | Material | n | Constitutive | m | x1 | y1 | x2 | y2 | ... |
|-------|----------|---|--------------|---|----|----|----|----|-----|
|-------|----------|---|--------------|---|----|----|----|----|-----|

Create a rigid block of material number n and constitutive number m. Defaults are n=1, m=1, if m, n omitted. Corner coordinates are: (x1,y1), (x2,y2) etc., in a clockwise direction. Continuation lines may be used but a pair of numbers defining a corner must not be separated. Only one BLOCK command may be used per run at present. Further blocks may be created with the SPLIT command, and unwanted ones deleted with the DELETE command. Any blocks may be changed to simply- or fully-deformable with the CHANGE command.

| Change | x1 | x2 | y1 | y2 | Sdef | Material | n | Constitutive | m |
|--------|----|----|----|----|------|----------|---|--------------|---|
| | | | | | Fdef | | | | |

All blocks with centroids lying within the range $x_1 < x < x_2$, $y_1 < y < y_2$ are changed to simply-deformable or fully deformable (Sdef or Fdef respectively). Material and constitutive numbers may also be changed.

| Damping | fcrit | freq | Mass | Stiffness | Internal |
|---------|-------|------|------|-----------|----------|
|---------|-------|------|------|-----------|----------|

Viscous damping is applied, in the form of Rayleigh damping. If a qualifier is not given as the third parameter, full damping is used, with fcrit as the fraction of critical damping, and freq as the centre frequency. The word "Mass" eliminates the stiffness-proportional dashpots, and "Stiffness" eliminates the mass-proportional dashpots.

The word "Internal" causes the specified damping to be applied to the 3 internal degrees of freedom of simply-deformable blocks.

DElete x1 x2 y1 y2

All blocks are deleted in the range
x1<x<x2, y1<y<y2

Dump n m

Dump memory to printer from the main array from address n to address m. Internal pointers MFREE, JUNK, IBPNT, ICPNT and IDPNT are also printed. MFREE gives the highest memory location that is currently free.

FRAction f

f is taken as the fraction of critical time-step to be used.

Fix x1 x2 y1 y2

All blocks are fixed in range
x1<x<x2 , y1<y<y2

FRee x1 x2 y1 y2

All blocks are set free in range
x1<x<x2 , y1<y<y2.
Note: by default, all blocks are free initially.

Generate x1 x2 y1 y2 Manual Gridpoints <glist> Zones <zlist>
Automatic

The first block encountered in the range x1<x<x2, y1<y<y2 is discretised as fully-deformable. The automatic option is not available yet. For manual generation, a list of grid-points, <glist>, and zones, <zlist> must be given. The format for <glist> is:

x1 y1 x2 y2 x3 y3 ,

where each x,y pair is a coordinate of a grid-point. If a given coordinate lies within a certain tolerance of a block corner, the grid-point is placed on that corner. If the coordinate lies within the same tolerance of a block edge, a new corner is created in the edge. The tolerance is taken as 0.9 times the rounding length. The format for <zlist> is:

11 m1 n1 12 m2 n2

Each triple corresponds to the three grid-points that define the zone, where the numbering of the grid-points refers to the order in <glist>, starting with the last (i.e. last grid-point is number 1).

Both <glist> and <zlist> may extend over an arbitrary number of continuation lines, but doubles and triples should not be split over two lines.

Gravity gx gy

Gravitational accelerations are set for the x- and y- directions.

Plot

All blocks and centroids are plotted

Print Blocks Contacts CORners Domains List DList

Data are printed on blocks, contacts, corners, domains and linked lists for blocks and domains.

| | | | |
|----------|------------|--------|------------|
| PROperty | Material n | Bulk b | Cohesion c |
| | n | K =b | Density d |
| | | KN=sn | Friction f |
| | | KS=ss | G=g |

Material properties are defined for material number n. Properties are: bulk modulus, b; shear modulus, g; density, d; joint normal stiffness, sn; shear stiffness, ss; friction coefficient, f; cohesion, c. The first parameter must be the specification of material number.

Restart

The program is restarted, using data from the restart file

RSet v ia ioff

The real value v is inserted in the main array at address ia, with offset ioff.

ROund d

Each block corner is rounded with a circle that is tangential to the two corresponding edges at a distance d from the corner.

SAve

The current problem state is saved on the restart file.

SCale s

Plot scale is set to s

SPlit x1 y1 x2 y2

All blocks in the path of a line extending from point (x1 y1) to (x2 y2) are split into two. At present, the line should not pass through any corner, or run too close to an existing edge.

STArt

The program does a cold start.

Stop

The run stops.

Appendix 1-2

Error numbers

- 1 Memory overflow
- 2 Unrecognisable command
- 3 Not start or restart as first command
- 4 Material number out of range
- 5 Block has less than 3 corners
- 6 Negative or zero block area
- 7 Missing parameter
- 8 Missing y-value
- 9 Constitutive number out of range
- 10 Unrecognised parameter
- 11 Contact stiffnesses undefined - cannot cycle
- 12 Zero mass block(s) present - cannot cycle
- 13 Mass damping for rigid-body motion too high.
- 14 Rounding length too great
- 15 Contact overlap too great.
- 16 Internal mass damping too high.
- 17 Cannot split fully-deformable block.
- 18 Only one block may be created at present -
 use SPLIT for more.
- 19 Cannot delete final contact in problem.
- 20 Internal error, subroutine DELC.
- 21 Cannot delete fully-deformable block at present.
- 22 Fifth parameter must be "Manual" or "Automatic"
- 23 Not available yet
- 24 Cannot find block in range
- 25 Zone pointer references non-existent grid-point
- 26 Missing data

APPENDIX II EXAMPLE RUNS

II-1 Ball Rolling

II-2 Collapse of Opening in Jointed Rock

II-1 Ball Rolling

This example illustrates that corner rounding can be taken to the extreme of creating a circle from an angular block. The complete input sequence to create the blocks and run the problem is given overleaf. Fig. II-1 shows the system of blocks after splitting, but before deletion of unwanted blocks and final rounding. Fig. II-2 is the plotted output from UDEC, with plots superimposed after successive 200-cycle increments.


```

START
PROP MAT=1 DENS=2000 KN=1.E08 KS=1.E08 F=2.
GRAVITY 4. -8.
DAMP =.5 16. MASS
FRAC 0.10
BLOCK 0.,-2. 0.,10.5 14.,10.5 14.,-2.
SPLIT -1.,.5 15.,8.5
SPLIT 10.5,5. 7.5,11.
SPLIT 12.5,6. 10.,11.
SPLIT -1.,3. 12.,9.5
DELETE 0.,9. 4.,7.
DELETE 0.,9. 6.,10.
DELETE 8.,11. 8.,11.
DELETE 11.,14. 7.,11.
FIX 0.,14. 0.,6.
ROUND 1.115
CYCLE 300
P B
PLOT 0
PROP MAT=1 F=0.80
DAMP 0.10 30 STIFFNESS
GRAVITY 0. -10.
CYCLE 200
P B
PLOT 1
CYCLE 200
P B
PLOT 1
CYCLE 200
P B
PLOT 1
CYCLE 200
P B
PLOT 1
CYCLE 200
P B
PLOT 1
SAVE
STOP
END

```

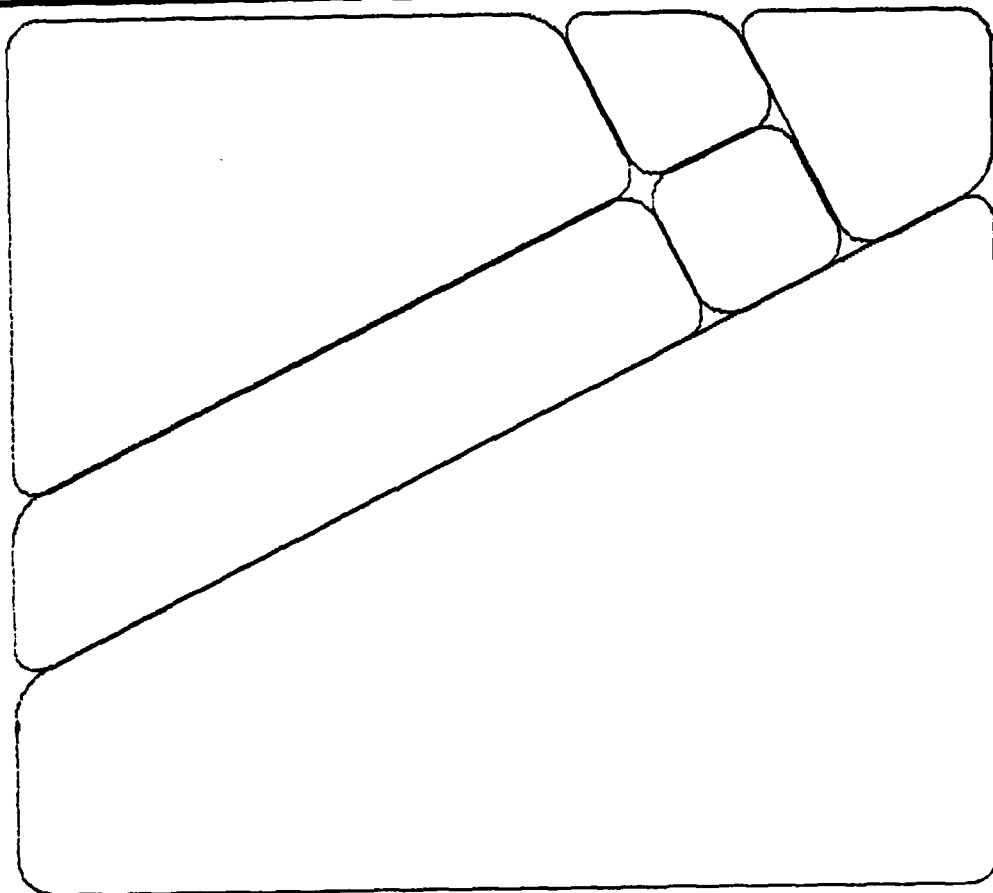


FIG. II-1: SYSTEM OF BLOCKS PRIOR TO DELETION

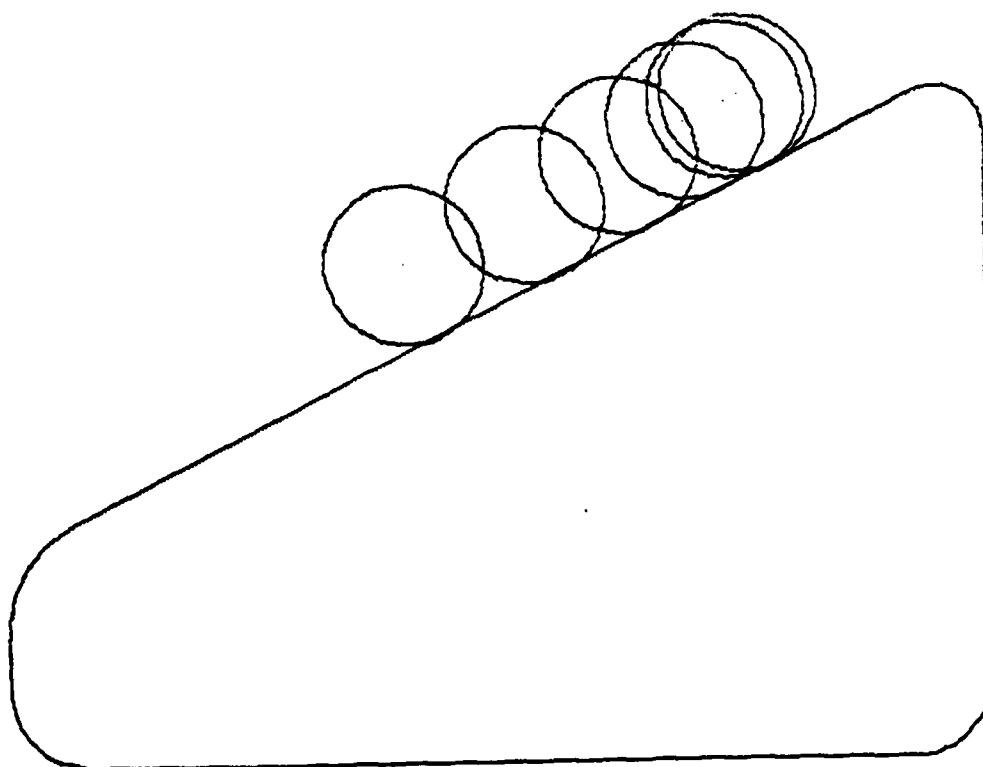
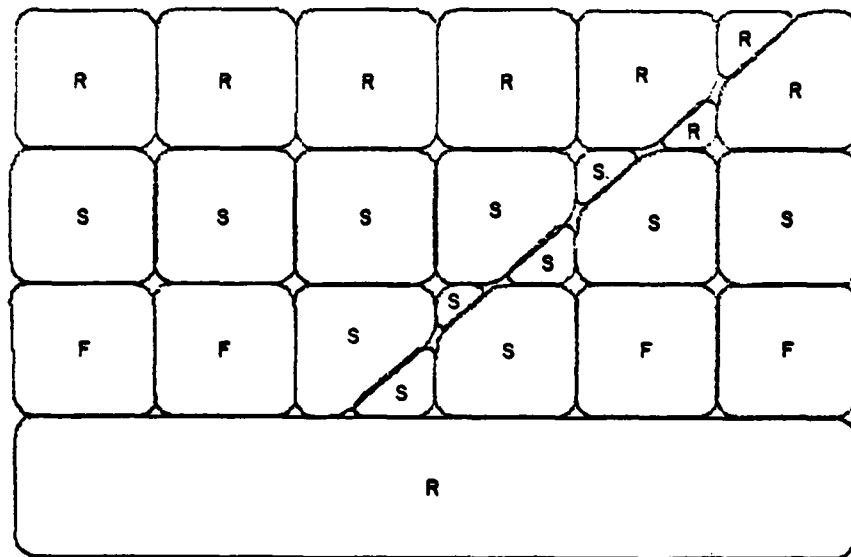


FIG. II-2: MOTION OF 'BLOCK' UNDER GRAVITY

II-2 Collapse of Opening in Jointed Rock

Figure II-3 shows the initial state of a rock system as produced by the input sequence given on the following page. The example is intended to demonstrate the interaction of all three block types in the same problem. The upper layer of rigid blocks is heavier than the rest. Figures II-4 onwards record the movements that take place after several blocks have been removed. It is interesting to note that "hang-ups" do not occur, due to the rounded corners. The fully-deformable blocks, which are discretised internally into four triangles, are deforming in modes that are more complex than those that are possible with simply-deformable blocks.



KEY:
R = RIGID
S = SIMPLY DEFORMABLE
F = FULLY DEFORMABLE

FIG.II-3: INITIAL GEOMETRY

```

START
SCALE .20
* SMALL TEST OF FINAL RUN
ROUND 1.5
FRAC .050
GRAVITY 0 -10
BLOCK 0 0 0 40 60 60 60 0
PROP MAT=1 DENS=2000 K=1.0E8 KS=1.0E8 BULK=1.0E7 G=0.2E7 F=0.5
PROP MAT=2 DENS=5000 K=1.0E8 KS=1.0E8 F=0.5
SPLIT -1 10 61 10
SPLIT -1 20 61 20
SPLIT -1 30 61 30
SPLIT 10 9 10 41
SPLIT 20 9 20 41
SPLIT 30 9 30 41
SPLIT 40 9 40 41
SPLIT 50 9 50 41
SPLIT 20 7.0 60 43.5
CHANGE 0 60 30 40 MATERIAL=2
CHANGE 0 60 20 30 SDEF
CHANGE 0 20 10 20 FDEF
CHANGE 20 40 10 20 SDEF
CHANGE 40 60 10 20 FDEF
FIX 0 60 0 10
GENERATE 0 10 10 20 MANUAL GRIDPOINTS (5,15) (0,20) (0,10)
+ (10,10) (10,20) ZONES 1,2,5 2,3,5 3,4,5 4,1,5
GENERATE 10 20 10 20 * G 15,15 10,20 10,10 20,10
+ 20,20 ZONES 1,2,5 2,3,5 3,4,5 4,1,5
GENERATE 40 50 10 20 * G 45,15 40,20 40,10 50,10
+ 50,20 Z 1,2,5 2,3,5 3,4,5 4,1,5
GENERATE 50,60 10,20 * G 55,15 50,20 50,10 60,10
+ 60,20 Z 1,2,5 2,3,5 3,4,5 4,1,5
DUMP 1 1
DAMP .25 .2
DAMP 0.25 1.0 INTERNAL
CYCLE 100
PLOT 0
P B
STOP
END

```

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FORWARDED TO EDC

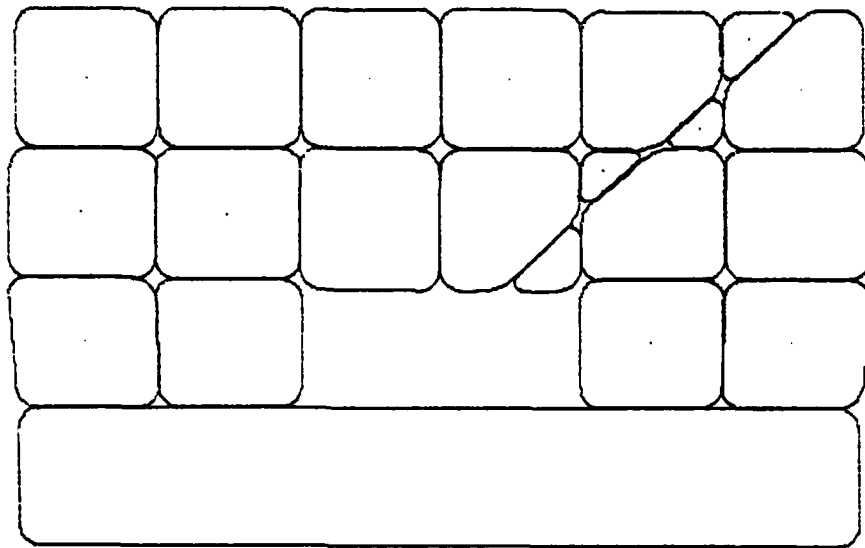


FIG. II - 4

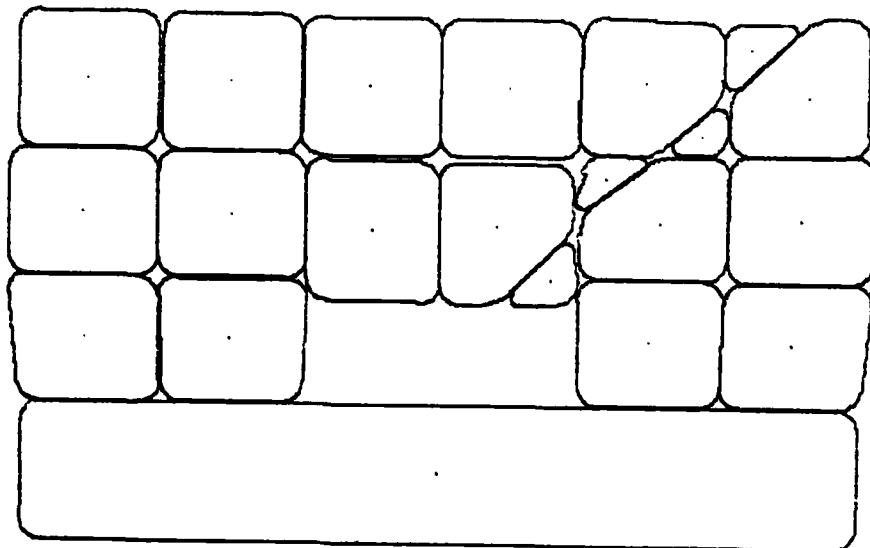


FIG. II - 5

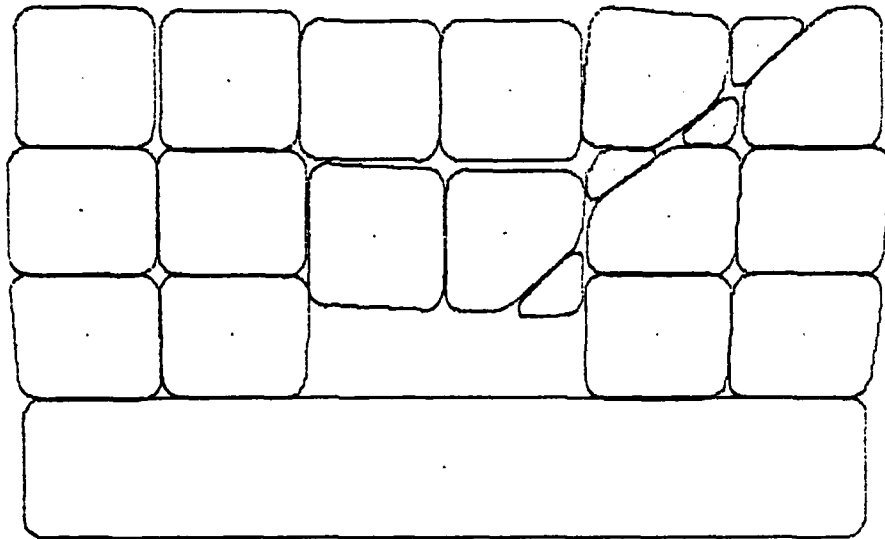


FIG. II-6

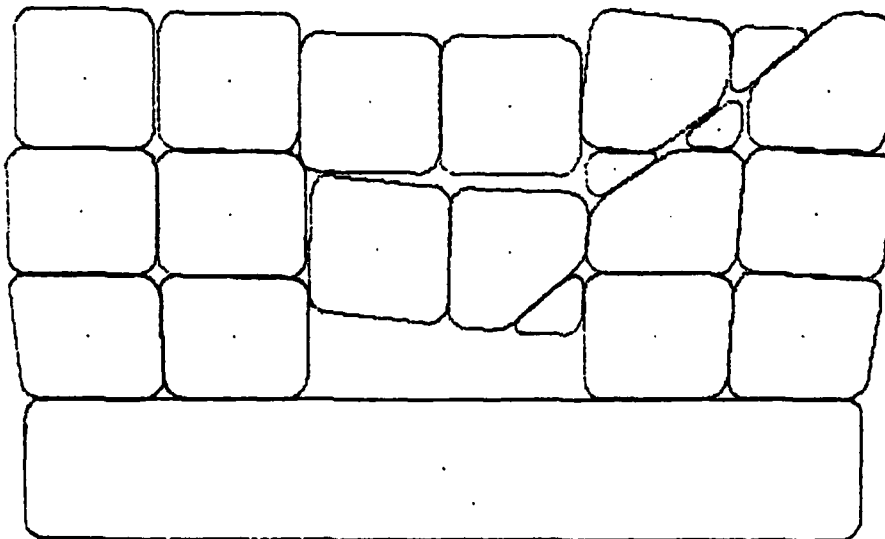


FIG. II-7

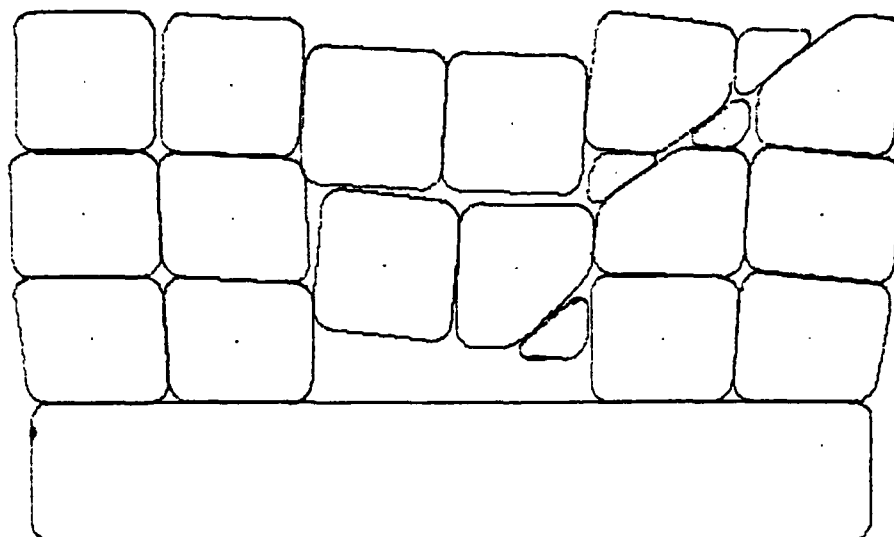


FIG. II - 8

APPENDIX III - PROGRAM GUIDE

III-1 Main Common Block Variables

III-2 Parameters & Data Groups

III-3 Subroutine Functions

III-4 Subroutine Calling Map

Appendix III-1 - Main Common Block Variables

| | |
|-----------|---|
| LINE(80) | Buffer for current input line in A1 format. |
| LINE1(80) | Buffer for next input line. |
| LPNT(1) | Pointer to start of parameter 1 in LINE() after removal of blanks, etc. |
| RAFLAG | |
| PPFLAG | .TRUE. if pore-pressure calculation requested |
| ERFLAG | .TRUE. if an error has occurred |
| STFLAG | .TRUE. if the first input line has been processed |
| COFLAG | .TRUE. if the current line is a continuation |
| NCFLAG | .TRUE. if the next line is a continuation |
| JMP SAV | Index of last computed GOTO in MON |
| NERR | Error number |
| JUNK | Pointer to list of spare memory groups |
| MFREE | First unused memory address |
| IBLOCK | Current block number |
| IDOM | Current domain number. |
| ISTACK | Stack pointer |
| NCYC | Currently requested number of cycles |
| NCTOT | Total number of cycles |
| TDEL | Time-step |
| FRAC | Requested fraction of critical time-step |
| IROUTE | Routing number, used in UDC |
| NLINE | Output line count |
| NPAGE | Output page count |
| JMP GEN | Routing number for continuation line in GEN |
| ALPHA | Mass damping coefficient |
| BETA | Stiffness damping coefficient |
| CON1 | Damping factor |
| CON2 | Damping factor |
| ALPB | Internal mass damping coefficient for simply deformable blocks. |
| C1B | Damping factor derived from ALPB. |
| C2B | " " |
| DEGRAD | $\pi/180$ |
| PI | 3.14159 |
| PSCALE | Plotting scale |
| ATOL | Distance between particles at which a contact is first formed. |
| BTOL | Distance between particles at which a contact is broken. |
| CTOL | Maximum (negative) overlap allowed when forming contacts |
| DTOL | Rounding length |
| DTOL2 | $=DTOL/2.0$ (maximum contact overlap) |
| ETOL | Limit on maximum domain displacement to trigger contact update. |
| FTOL | |
| GTOL | |
| HTOL | |

| | |
|----------|--|
| IBPNT | Pointer to list of blocks |
| ICPNT | Pointer to list of contacts |
| IDPNT | Pointer to list of domains |
| IODPNT | Pointer to outer domain |
| AKN(I) | Normal joint stiffness, material I |
| AKS(I) | Shear joint stiffness, material I |
| AMU(I) | Joint friction coefficient, material I |
| COH(I) | Joint cohesion, material I |
| DENS(I) | Density, material I |
| BULK(I) | Bulk modulus, material I |
| SHEAR(I) | Shear modulus, material I |
| ALAM1(I) | Lame constant, material I |
| ALAM2(I) | Lame constant, material I |
| IA() | Main array |

Appendix III-2 Parameters & Data Group

Offsets for block data array

Note: the first integer in each block array
---- (offset 0) is the block type number, as follows:
1 rigid block
2 simply-deformable block
3 fully-deformable block

| | |
|-------|--|
| KB | Pointer to next block in block list |
| KP | Pointer to one corner in block's corner list |
| KMAT | Material number |
| KCOMS | Constitutive number |
| KX | x coordinate of centroid |
| KY | y coordinate of centroid |
| KXD | x velocity |
| KYD | y velocity |
| KTD | Angular velocity (anticlockwise positive) |
| KBM | Block mass |
| KBI | Moment of inertia |
| KBFX | x centroid force-sum |
| KBFY | y centroid force-sum |
| KBFT | centroid moment sum |
| KBEX | extension pointer (to SDEF or FDEF data) |

Offsets for corner data array

Note: the first integer (offset 0) contains
---- the value MCOR to denote a corner

| | |
|------|---|
| KL | Pointer to next corner or contact on block, in clockwise direction. |
| KR | Pointer to next corner in anticlockwise direction |
| KNB | Pointer to host block |
| KXP | x coordinate of corner |
| KYP | y coordinate of corner |
| KXCP | x coordinate of local circle centre |
| KYCP | y coordinate of local circle centre |
| KRAD | Radius of local circle |
| KXDP | x velocity of corner |
| KYDP | y velocity of corner |
| KGP | Pointer to corresponding grid-point if block is fully-deformable |

Offsets for contact data array

Note: the first integer (offset 0) contains
the value MCON, to denote a contact

| | |
|-------|---|
| KC | Pointer to next contact in contact list |
| KB1 | Address of first block involved in contact |
| KB2 | Address of second block involved in contact |
| KL1 | Pointer to next item in clockwise list of block corresponding to KB1 |
| KL2 | same as KL1, but for block KB2 |
| KD1 | Address of domain to left of contact, going from block KB1 to KB2 |
| KD2 | Address of domain to right of contact going from block KB1 to KB2 |
| KCM | Material type number |
| KCC | Constitutive number |
| KXC | x contact coordinate |
| KYC | y contact coordinate |
| KXDC | relative x velocity (of block KB2 relative to block KB1) |
| KYDC | relative y velocity |
| KCS | relative shear displacement |
| KCN | relative normal displacement |
| KCFS | shear force |
| KCFN | normal force (positive compression) |
| KCCOD | code number: 1 corner/corner contact 2 corner/edge contact (KB1 .. corner, KB2 .. edge) 3 edge/corner contact |

Offsets for domain data array

Note: the first integer (offset 0) contains the
---- value MDOM, to denote a domain

| | |
|--------|---|
| KD | Pointer to next domain in domain list |
| KPP | Pore-pressure for domain |
| KUMAX | Fictitious domain displacement |
| KDLOOP | Pointer to one contact in anticlockwise list around domain |

Simply-deformable extension array

KED11)
KED12) Strain-rate
KED21) tensor
KED22)

KSI11)
KSI12) Internal stress
KSI21) tensor
KSI22)

KSA11) Applied stress
KSA12) tensor (multiplied
KSA21) by block
KSA22) area)

Offsets for grid-point data

KG Pointer to next grid-point in grid-point list
XCOR Pointer to corresponding block corner
KXG x coordinate
KYG y coordinate
KXDG x velocity
KYDG y velocity
KGFX x force-sum
KGFY y force-sum
KGPM grid-point mass

Offsets for zone data

KZ Pointer to next zone in zone list
KZG Start of triple pointer to 3 surrounding
grid-points
KZS11)
KZS12) Stress tensor
KZS22)
KZM Zone mass

Logical unit numbers

LUNIF Unit number for input file
LUNOF Unit number for output file
LUNG Unit number for general I/O (e.g. restart)
LUNP Unit number for plotted output

Number of words in data arrays

NVCR Corner
NVBL Block
NVCN Contact
NVDO Domain
NVSD Simply-deformable extension
NVZO Zone
NVGP Grid-point

Array limits

MTOP Size of main array (IA)
NMAT Maximum number of materials
NCONS Maximum constitutive numbers
NTYP Number of block types (rigid, SDEF etc)

Head codes (contents of first integer in data array)

MRIG =1 Rigid block
MSDEF =2 Simply-deformable block
MFDEF =3 Fully-deformable block
MCON Corner
MCON Contact
MDOM Domain

Appendix III-3 Subroutine Functions

SUBROUTINE ACCUM(A)

-RECORD MAX AND MIN VELOCITIES FOR CORNER

SUBROUTINE APLDT

-BLOCKS ARE PLOTTED

SUBROUTINE ARC(RAD)

-PLOT ARC, RADIUS RAD, CENTRE (XP,YP), FROM (X1,Y1)
-TO (X2,Y2). PEN ASSUMED DOWN ON ENTRY.

BLOCK DATA FRED

-INITIALISE FIXED PARAMETERS. THIS ROUTINE
-(AND THE COMMON BLOCK /PARAM/) CAN BE REPLACED
-BY A SET OF PARAMETER STATEMENTS IF
-SUPPORTED BY THE COMPILER.

SUBROUTINE BRIDGE(IAD1,IAD2,IAB1,IAB2,ICOD,IDNEW)

-MAKE A CONTACT TO BRIDGE DOMAIN IDOM BETWEEN LIST ITEMS
-IAD1 AND IAD2 (BLOCKS IAB1 & IAB2). ICOD=CODE OF NEW SEG.
-(XC,YC)=COORDINATES OF CONTACT.
-IDNEW=RETURNED NEW DOMAIN NUMBER.

SUBROUTINE CC(A1,A2)

-PARAMETERS FOR CORNER-CORNER CONTACT

SUBROUTINE CE(A1,A2,AP)

-PARAMETERS FOR CORNER-EDGE CONTACT

SUBROUTINE CHANGE

-CHANGE ATTRIBUTES OF EXISTING BLOCKS WITHIN RANGE

SUBROUTINE CL1

-ELASTIC, ISOTROPIC CONSTITUTIVE LAW

SUBROUTINE COORD(IAC)

-UPDATE COORDINATES FOR CONTACT IAC

SUBROUTINE CRAD(IAB)

-CORNER RADIUS AND CIRCLE CENTRE FOR EACH
-CORNER OF BLOCK IAB

SUBROUTINE CREATE

-CREATE A BLOCK (FROM BLOCK COMMAND)

SUBROUTINE CVEL(B,C)

-UPDATE RELATIVE CONTACT VELOCITIES
-B() IS BLOCK ARRAY; C() IS CONTACT ARRAY

THIS PAGE IS BEST QUALITY PRINTING
FROM COPY FURNISHED TO BPG

SUBROUTINE CVELFD(C)

-RELATIVE CONTACT VELOCITIES FROM G.P. VELOCITIES

SUBROUTINE CYCLE

-MAIN CALCULATION CYCLE

SUBROUTINE DAMP

-PROCESS DAMPING COMMAND

SUBROUTINE DELB

-DELETE BLOCK IBLOCK

SUBROUTINE DELC(IAC)

-DELETE CONTACT IAC

SUBROUTINE DELLST(IAD,IPNT,KOFF)

-DELETE ITEM IAD FROM LIST POINTED TO

-BY IPNT, WITH OFFSET WITHIN AN ITEM OF KOFF

SUBROUTINE FDC(IPP)

-DEAL WITH FORCE APPLIED AT CORNER OF FULLY-DEF BLOCK

SUBROUTINE FDE(IPA,IPB)

-DEAL WITH FORCE APPLIED TO EDGE OF FULLY-DEF BLOCK

SUBROUTINE FDEF

-SCAN THROUGH GRID-POINTS & ZONES FOR FULLY-DEF BLOCK

SUBROUTINE FIND(N,NG)

-FIND N WORDS OF MEMORY

-NG = RETURNED ADDRESS

-ERFLAG IS SET IF MEMORY CANNOT BE FOUND

SUBROUTINE FIX(IFIX)

-SET OR RESET FIX FLAG FOR ALL BLOCKS WITHIN RANGE

SUBROUTINE FORD(CON)

-FORCE/DISP CALC. FOR CONTACT

SUBROUTINE GEN

-PROCESS "GENERATE" COMMAND, WHICH GENERATES ZONING FOR

-FULLY-DEFORMABLE BLOCK EITHER MANUALLY OR AUTOMATICALLY.

SUBROUTINE GEU(IAB,AR,RG2)

-FOR BLOCK IAB, COMPUTES AREA, RADIUS OF

-GYRATION SQUARED AND CENTROID (XP,YP)

FUNCTION GETR(A,KOFF)

-GET REAL VALUE FROM MAIN ARRAY WITH

-OFFSET KOFF

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

SUBROUTINE HALT

- ERROR MESSAGES PRINTED HERE
- PROGRAM STOPS UNDER CERTAIN CONDITIONS

FUNCTION ICALB(IC)

- GET CALLING ADDRESS FOR CONTACT IC FROM BLOCK IBLCK

FUNCTION IDUMP(IAD)

- RETURN DOMAIN ADDRESS FOR SEGMENT IAD, BLOCK IBLCK

FUNCTION IGET(IAD,KOFF)

- GET INTEGER FROM MAIN ARRAY @ IAD+KOFF

FUNCTION IGETGP(I)

- FIND I'TH GRIDPOINT ADDRESS, BLOCK IBLCK

SUBROUTINE IMSG

- INITIAL MESSAGE FROM CODE

SUBROUTINE INERT(X0,Y0,X,Y,YP,YS,RM1CAX)

- COMPUTE MOMENT OF INERTIA ABOUT GIVEN AXIS

SUBROUTINE INI

- PREPARE FOR CYCLING

SUBROUTINE INSECT(YES)

- FIND IF 2 LINES INTERSECT (RETURNS YES=.TRUE.)
- AND IF SO, RETURN COORDS (XP,YP).
- THE 2 LINES ARE ((X1,Y2),(X2,Y2))
- AND ((X3,Y3),(X4,Y4))

SUBROUTINE IO(I,NPAR)

- ALL NON-STANDARD I/O DONE HERE

FUNCTION IVAR(NPAR)

- TO RETURN INTEGER VALUE OF PARAMETER NPAR

SUBROUTINE JUMPB

- TO JUMP TO NEW BLOCK IN DOMAIN FROM CONTACT IAD

SUBROUTINE LOSE(N,NG)

- TO RETURN N WORDS OF MEMORY AT ADDRESS NG

SUBROUTINE MAKEB(IAB,UMAS,UMOI,X,Y)

- TO INSTALL MASS, MOI & CENTROID
- IN BLOCK IAB

SUBROUTINE MAKEC(IAC,IAB1,IAB2,IL1,IL2,ID1,ID2,ICOD)

- FILL CONTACT DATA BLOCK AT IAC, AND LINK TO CONTACT LIST.
- IAB1, IAB2 = BLOCK ADDRESSES
- IL1, IL2 = ASSOCIATED CIRCULATING LISTS
- ID1, ID2 = DOMAIN ADDRESSES
- ICOD = CONDITION CODE

THIS PAGE IS BEING REPRODUCED FROM COPY FURNISHED TO BDO

SUBROUTINE MAKEP(IAD,IAL,IAR,X,Y,IAB)
 -FORM CORNER @ IAD, WHERE IAL=CLOCKWISE LINK,
 -IAR=REVERSE LINK, (X,Y)=CORNER COORDS,
 -IAB=BLOCK ADDRESS

SUBROUTINE MATCH(NTAB,INDEX,NPAR,JUMP,BAD)
 -TO MATCH INPUT STRING TO KEYWORD IN TABLE
 -INPUT: NTAB TABLE OF KEYWORDS
 - INDEX LIST OF LENGTHS
 - NPAR PARAMETER NO. IN INPUT LINE
 -OUTPUT: JUMP DISPATCH NUMBER
 - BAD .TRUE. FOR MISSING PARAMETER
 - OR STRING NOT FOUND.

SUBROUTINE MUN
 -MONITOR

SUBROUTINE MOTION(A)
 -LAW OF MOTION FOR SINGLE BLOCK

SUBROUTINE MOVE(IAD1,IAD2,NWD)
 -MOVE NWD WORDS FROM ADDRESS IAD1 TO IAD2

SUBROUTINE MVBK(IAC,IAB,ICOD)
 -MOVE CONTACT BACKWARDS PAST CORNER (TAKING
 -WITH IT ALL INTERMEDIATE CONTACTS).
 -IAC=CONTACT; IAB=BLOCK; ICOD=NEW CODE.

SUBROUTINE MVFOR(IAC,IAB,ICOD)
 -MOVE CONTACT FORWARDS PAST CORNER (TAKING WITH IT
 -ALL INTERMEDIATE CONTACTS).
 -IAC=CONTACT; IAB=BLOCK; ICOD=NEW CODE

SUBROUTINE NEXTCP(IAD,IADN,ICALL)
 -FIND NEXT ITEM (IADN) IN CORNER CHAIN AND
 -ITS CALLING ADDRESS, GIVEN CURRENT ITEM, IAD
 -BLOCK NUMBER ASSUMED TO BE IBLOCK

SUBROUTINE NEXTD(IAD,IADN)
 -FIND NEXT ITEM (IADN) IN DOMAIN CHAIN.
 -IAD=CURRENT ITEM; IDOM=DOMAIN.
 -IBLOCK IS UPDATED FOR EACH CONTACT ENCOUNTERED.

SUBROUTINE NEXTDS
 -GET NEXT SEGMENT IN DOMAIN, WHERE SEGMENT IS
 -DEFINED AS CORNER OR EDGE. IAS IS SEGMENT NUMBER.
 -CORNER=.TRUE. IF CURRENT SEGMENT IS A CORNER,
 -FALSE OTHERWISE. CONTAC=.TRUE. IF A CONTACT
 -HAS JUST BEEN PASSED THROUGH.
 -"ITEM" COUNTS CONTACTS+SEGMENTS

THIS PAGE IS ONLY QUALITY FRACITABLE
 FROM OUR FRICTION TO DEC

SUBROUTINE NEXTP(IAD,IAP)
 -FIND NEXT CORNER (IAP) IN BLOCK CHAIN,
 -GIVEN CURRENT ITEM, IAD, IN CHAIN.
 -BLOCK NUMBER IS IBLOCK

SUBROUTINE OUTCP(IAD)
 -PRINT LIST ITEM IAD

SUBROUTINE OUTFG(IAD1)
 -SCAN THROUGH GRID-POINTS FOR PRINTOUT

SUBROUTINE OUTFDZ(IAD1)
 -SCAN THROUGH ZONE FOR PRINTOUT

SUBROUTINE OUTG(A,IAD,ICOR)
 -PRINT DATA FOR GRID-POINT, ARRAY A()

SUBROUTINE OUTSD(A)
 -PRINT OUT INTERNAL VARIABLES FOR SIMPLY-DEF. BLOCK

SUBROUTINE PLOTS(X,Y,I)
 -SCALED PLOT

SUBROUTINE POP(ITEM)
 -POP ITEM FROM STACK

SUBROUTINE PRINT
 -MAIN PRINTOUT IS DONE HERE

SUBROUTINE PROP
 -RESPOND TO "PROPERTY" COMMAND

SUBROUTINE PUTFCE(B)
 -PUT FORCES FROM CONTACT INTO BLOCK AT B()

SUBROUTINE PUTFDF(A,FX,FY)
 -ADD FORCES INTO GRID-POINT'S FORCE SUM

SUBROUTINE PUTI(IAD,KOFF,I)
 -PUT INTEGER I AT IA(IAD+KOFF)

SUBROUTINE PUSH(ITEM)
 -PUSH ITEM ONTO STACK

SUBROUTINE PUTP(ARRAY,I)
 -PUT VALUE OF PARAMETER I IN ARRAY
 -AND CHECK FOR RANGE OF I

SUBROUTINE PUTR(A,KOFF,V)
 -INSERT REAL VALUE V INTO MAIN ARRAY
 -WITH OFFSET KOFF

THIS PARTITION IS QUANTITATIVELY PRACTICABLE
FOR A CORNER POINT OF 10000

SUBROUTINE PUTSTR(B)
-PUT APPLIED STRESSES IN SIMPLY-DEFORMABLE BLOCK

SUBROUTINE PVEL(B,C)
-UPDATE CORNER VELOCITIES AND COORDINATES.
-B() IS BLOCK ARRAY; C() IS CORNER ARRAY.

SUBROUTINE PVELFD(A)
-UPDATE CORNER CUORDS & VELOCITIES FROM G.F. VALUES

LOGICAL FUNCTION QBC(IAB)
-RETURNS .TRUE. IF BLOCK IAB HAS ANY CONTACTS

SUBROUTINE QUERY(IAC)
-CORNER IS SET .TRUE. IF CONTACT IAC ON
-BLOCK IBLOCK CORRESPONDS TO A CORNER.

LOGICAL FUNCTION RANGE(IAB)
-.TRUE. IF BLOCK CENTROID LIES OUTSIDE RANGE GIVEN BY
-USER.

SUBROUTINE REL1
-COORDS OF POINT (XP,YP) RELATIVE TO
-LINE ((X1,Y1),(X2,Y2)). COMPUTES ALSO
-LENGTH (Z), COS (C) & SIN (S) OF LINE.
-OUTPUT COORDS: (XR,YR)

SUBROUTINE REL2
-SAME AS REL1, BUT C AND S ARE ASSUMED AS INPUT

FUNCTION RVAR(NPAR)
-TO RETURN REAL VALUE OF PARAMETER NPAR

LOGICAL FUNCTION SEP(ICHAR)
-RETURNS .TRUE. IF ICHAR IS A SEPARATOR

SUBROUTINE SPLIT
-SPLIT BLOCKS IN THE PATH OF GIVEN LINE

SUBROUTINE SPLI(IAD1,IAD2,XA,YA,XB,YB)
-SPLIT BLOCK (IBLOCK) BETWEEN (XA,YA) AND (XB,YB)
-IAD1 AND IAD2 ARE LIST ITEMS JUST BEFORE
-SEGMENTS THAT WILL BE SPLIT

SUBROUTINE START
-STARTUP ASSIGNMENTS & INITIALISATION

SUBROUTINE STRAIN(B,E)
-COMPUTE STRAIN-RATE TENSOR FOR SIMPLY-DEFORMABLE BLOCK
-B() IS THE BLOCK ARRAY
-E() IS THE EXTENSION ARRAY FOR SIMPLY-DEF DATA

SUBROUTINE STRESS(B,E)
 -STRESSES FROM STRAIN-RATES ... SIMPLY-DEFORMABLE BLOCK
 -B() IS THE BLOCK ARRAY
 -E() IS THE SDEF EXTENSION ARRAY

SUBROUTINE TIDY
 -TO ELIMINATE BLANKS, ETC. FROM INPUT
 -LINE AND MAKE INDEX TO LOCATION OF PARAMETERS

SUBROUTINE TOLSET
 -SET TOLERANCES FROM AVERAGE BLOCK DIMENSIONS

SUBROUTINE TRUE(IAD,X,Y)
 -GET TRUE LIST ITEM BEFORE CUT @ (X,Y).
 -IAD IS NEAREST CORNER BEFORE (X,Y), AND
 -IAD IS OVERWRITTEN WITH NEW LIST ADDRESS.

PROGRAM UDEC
 -UNIVERSAL DISTINCT ELEMENT CODE

 - VERSION 1.0
 - -----
 -WRITTEN BY P.A.CUNDALL, MARCH 1960, FOR U.S. ARMY
 -(EUROPEAN RESEARCH OFFICE) AND DEFENSE NUCLEAR AGENCY
 -UNDER CONTRACT DAJA37-79-C-0548.

SUBROUTINE UPDATE
 -UPDATE DOMAIN IDOM

SUBROUTINE UPDT
 -UPDATE CONTACTS IN DOMAIN IDOM

SUBROUTINE VAR(NPAR)
 -COMMON ROUTINE FOR IVAR & RVAR

SUBROUTINE XYCOR(A,X,Y)
 -RETURNS CORNER COORDS (X,Y) FOR CORNER ARRAY A()

SUBROUTINE XYFD(A,X,Y,XD,YD)
 -RETRIEVE GRID-POINT COORDS & VELOCITIES

SUBROUTINE ZCS
 -TO RETURN LENGTH (Z) AND UNIT VECTOR (C,S)
 -OF LINE (X1,Y1),(X2,Y2)

SUBROUTINE ZSTRS(A,IZON)
 -STRAIN-RATES, STRESSES & HENCE G.P. FORCES FOR A ZONE

Appendix III-4 Subroutine Calling Map

```

UDC
-START      -IO
-
-MON        -TIDY      -SEP
           -MATCH
           -IO
           -IMSG
           -PROP      -MATCH
           -FIX
           -DAMP
           -DELB      -NEXTCP
                   -DELC      -NEXTD      -DELLST      -LOSE
                   -LOSE
                   -DELLST
           -CHANGE    -MATCH
                   -FIND
-
-HALT       -IO
-
-PRINT      -MATCH
           -OUTSD
           -OUTFDZ
           -OUTFDG    -OUTG
           -OUTCP
           -NEXTCP
           -NEXTD
           -NEXTP      -NEXTCP
-
-CREATE     -FIND
           -MATCH
           -GEO      -INERT
           -CRAD
           -TOLSET    -GEO
-
-SPLIT      -FIND
           -MOVE
           -GEO      -INERT
           -NEXTCP
           -NEXTP      -NEXTCP
           -MAKEP
           -MAKEC
           -MAKEB
           -CRAD
           -COORD    -NEXTP      -NEXTCP
                   -CC        -ZCS
                   -CE        -REL1
                               -ZCS
                               -REL2
           -TOLSET    -GEO
           -UPDATE    -UPDT      -JUMPB      -NEXTD      -NEXTCP
                               -NEXTP      -NEXTCP
                               -NEXTDS      -NEXTCP
                                       -QUERY
                                       ~JUMPB      -NEXTD
                                               -NEXTP      -NEXTCP
                               -NEXTP      -NEXTCP
                               -CE        -REL1
                                       -ZCS
                                       -REL2
                               -REL2
                               -NEXTCP
                               -CC        -ZCS
                               -BRIDGE    -FIND
                                       -NEXTCP
                                       -COORD      -NEXTP      -NEXTC
                                               -CC        -ZCS
                                               -CE        -REL1
                                               -ZCS
                                               -REL2
-
-PUSH       -FIND
-
-APLOT      -PLOTS
           -ARC      -PLOTS
-
-INI        -CRAD
           -GEO      -INERT
           -MAKEB
           -TOLSET    -GEO

```

| | | | |
|--------|---------|---------|----------------|
| -CYCLE | -MOTION | -CVEL | |
| | | -PVEL | |
| | -STRAIN | -XYCOR | |
| | -STRESS | -CL1 | |
| | | -CL2 | |
| | | -CL3 | |
| | | -CL4 | |
| | | -CL5 | |
| | -FDEF | -GPMOT | |
| | | -XYFD | |
| | | -PVELFD | |
| | | -NEXTCP | |
| | | -ZCS | |
| | | -REL2 | |
| | | -CVELFD | |
| | | -ZSTRS | -XYFD |
| | -FORD | -NEXTP | -NEXTCP |
| | | -CC | -ZCS |
| | | -CE | -REL1 |
| | | | -ZCS |
| | | | -REL2 |
| | | -DELC | -NEXTD |
| | | | -DELLST |
| | | | -LOSE |
| | | -CLJ2 | |
| | | -CLJ3 | |
| | | -CLJ4 | |
| | | -CLJ5 | |
| | | -PUTFCE | |
| | | -PUTSTR | |
| | | -FDC | -PUTFDF |
| | | -FDE | -PUTFDF |
| | | -MVFOR | -NEXTP -NEXTCP |
| | | | -NEXTCP |
| | | -MVBK | -NEXTCP |
| | | | -NEXTP -NEXTCP |
| | -ACCUM | | |
| | -NEXTP | *NEXTCP | |
| | -NEXTD | | |
| | -UPDATE | | |
| -GEN | -FIND | | |
| | -MATCH | | |

Most "functions" and the commoner subroutines which are listed below have been left out of the "calling map":

Functions

Subroutines

GETR
ICALB
IDOMP
IGET
IGETCP
IVAR
QBC
QTEST
RANGE
RVAR

PUTI
PUTP
PUTR
VAR

APPENDIX IV - GUIDE TO PROGRAM CHANGES

APPENDIX IV - Guide to Program Changes

IV-1 Parameters

All Fortran variables initialised in the BLOCK DATA subroutine FRED are constant during a UDEC run, and the DATA statements may be replaced with PARAMETER statements if these are supported by the compiler. This should lead to an increase in efficiency.

IV-2 Storage of Integers and Real Numbers

Program UDEC, as written, assumes that a real Fortran variable occupies the space of two integers. This correspondence is only important for quantities stored in the main array IA(). If UDEC is to be run on a machine having a different convention for storing variables, the offsets defined in the BLOCK DATA subroutine will need to be changed. The offset of a variable, referred to a particular data array, is defined as the number of words from the start of the data array to the location of the variable, where a "word" can be integer or real, depending on the type of variable. The numbering for integers starts at 0, and the numbering for reals starts at 1. As an example, the offsets are given below for the corner data array for the standard program UDEC, and for the program as it would be set up on a machine in which integers occupy the same space as reals.

| <u>Variable</u> | For 2 integers ≡ 1 real | | For 1 integer ≡ 1 real | |
|-----------------------|-------------------------------|---------------|------------------------------|---------------|
| | INTEGER COUNT | REAL COUNT | INTEGER COUNT | REAL COUNT |
| <head code> (integer) | 0 | (1) | 0 | (1) |
| KL (integer) | 1 | | 1 | (2) |
| KR " | 2 | (2) | 2 | (3) |
| KNB " | 3 | | 3 | (4) |
| KXP (real) | (4) | 3 | (4) | 5 |
| KYP " | (6) | 4 | (5) | 6 |
| KXCP " | (8) | 5 | (6) | 7 |
| KYCP " | (10) | 6 | (7) | 8 |
| KRAD " | (12) | 7 | (8) | 9 |
| KXDP " | (14) | 8 | (9) | 10 |
| KYDP " | (16) | 9 | (10) | 11 |
| KGP (integer) | (18) | (10) | 11 | (12) |

data array
for corners
(see Appendix III-2)

numbers not in parentheses are
offsets used in program; other numbers
are included to show the full sequence.

IV-3 Non-standard I/O and other Operations

All non-standard input/output is done in subroutine IO, and consists of OPEN and CLOSE calls for the input and output files. These calls may be replaced by equivalents if another computer is to be used.

The DECODE statements in subroutines IVAR and RVAR may be non-standard on some machines, and should be replaced as necessary. No other ENCODE or DECODE statements are used in the program.

Subroutine VAR packs characters into array IBUF, which is declared a BYTE array. On some machines this operation may have to be done with an ENCODE statement, if BYTE or INTEGER*1 variables are not allowed.

All INTEGER*2 statements may be omitted if they refer to single variables; statements that refer to arrays should be replaced by DIMENSION statements.

The statement INCLUDE 'COMMON.FTN' inserts at that point in the program the parameter blocks and main common block. Many computers have a similar facility, but is invoked differently.

IV-4 User-supplied Constitutive Subroutines

Dummy subroutines CL2, CL3, CL4 and CL5 may be replaced by real subroutines: these will be called for materials with constitutive numbers 2,3,4, and 5 respectively. Input and output variables are passed in common block /CLCOM/, where the names of the variables have the following meaning:

| | | |
|--------|--------------------------|--|
| INPUT | { S11 S12 S22 } | components of current stress tensor |
| | | |
| | { DE11 DE12 DE22 } | components of strain increment tensor |
| | | |
| OUTPUT | { DS11 DS12 DS22 } | components of stress increment tensor |
| | | |